



NATURAL

Natural X

Version 5.1.1 for Windows

Version 3.1.6 for OS/390

Version 5.1.1 for UNIX

 **SOFTWARE AG**



This document applies to Natural Version 5.1.1 for Windows, Version 3.1.6 for OS/390, Version 5.1.1 for UNIX and to all subsequent releases. Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

© June 2002, Software AG
All rights reserved

Software AG and/or all Software AG products are either trademarks or registered trademarks of Software AG. Other products and company names mentioned herein may be the trademarks of their respective owners.

Table of Contents

NaturalX - Overview	1
NaturalX - Overview	1
Introduction to NaturalX	2
Introduction to NaturalX	2
Why NaturalX?	2
Programming Techniques	3
Object-Based Programming	3
Defining Classes	3
Defining Interfaces	3
Interface Inheritance	4
Installing NaturalX	5
Installing NaturalX	5
Installing NaturalX under Windows 98/NT/2000	5
Installing NaturalX under UNIX	5
Prerequisites	5
Installation Procedure	6
NaturalX System Architecture under OS/390	7
Overview	7
Environment Variables	10
NaturalX Server Front-End	15
NaturalX Output Files	15
The NaturalX Server Monitor	16
The DCOM Buffer Pool	16
Installing NaturalX under OS/390	16
Developing NaturalX Applications	17
Developing NaturalX Applications	17
Using the Class Builder	17
Defining Classes	17
Creating a Natural Class Module	17
Specifying a Class	17
Defining an Interface	18
Assigning an Object Data Variable to a Property	18
Assigning a Subprogram to a Method	18
Implementing Methods	18
Using Classes and Objects	20
Defining Object Handles	21
Creating an Instance of a Class	21
Invoking a Particular Method of an Object	21
Accessing Properties	21
Sample Application	23
*THIS-OBJECT System Variable	23
Distributing NaturalX Applications	25
Distributing NaturalX Applications	25
General	25
Internal, External and Local Classes	25
Globally Unique Identifiers - GUIDs	26
Using the Class Builder	27
Using the Data Area Editor	27
NaturalX Servers	27
COM Classes and Servers	28
NaturalX Classes and Servers	28
NaturalX Servers and Natural Sessions under OS/390	28
NaturalX Servers and Natural Sessions under Windows 98/NT/2000 and UNIX	30

The Role of the Server ID	30
Organizing Server IDs	32
Activation Policies	33
Activation Policies Under Windows 98/NT/2000 and UNIX	33
Activation Policies Under OS/390	34
Setting Activation Policies	34
When to use which Activation Policy	35
Registration	38
Registration with Natural	38
Automatic Registration	38
Manual Registration	39
Registration Files and Type Library	42
Client Registration	42
Registration Hints	43
DCOMPARM System Command - OS/390 Only	45
server-ID	45
Type Information	45
Overview	45
NaturalX and Type Information	46
Using Type Information	46
Configuration Overview	49
Server Configuration - General Settings	49
Server Configuration - Application-Specific Settings	50
Client Configuration - General Settings	51
Client Configuration - Application-Specific Settings	51
Sample Application	51
Security with NaturalX	52
Security with NaturalX	52
Overview	52
Activation Security	52
Applications	52
Authorizations using the Registry	53
Call Security	53
Authorizations using Natural Security	53
Security Hints and Suggestions	55
NaturalX Configuration Examples	56
NaturalX Configuration Examples	56
DCOM Configuration on Windows NT/2000	56
Configuring NaturalX Servers on Windows NT/2000	57
Configuring NaturalX Clients on Windows NT/2000	66
DCOM Configuration on Windows 98	69
Configuring NaturalX Servers on Windows 98	70
Configuring NaturalX Clients on Windows 98	74
DCOM Configuration on Windows 98 in a Windows NT Domain	77
Configuring NaturalX Servers on Windows 98 in a Windows NT Domain	78
Configuring NaturalX Clients on Windows 98 in a Windows NT Domain	84
DCOM Configuration on UNIX with EntireX	87
Configuring NaturalX Servers on UNIX	87
Configuring NaturalX Clients on UNIX	89
DCOM Configuration on OS/390	89
NaturalX System Registry Entries	90
NaturalX System Registry Entries	90
Registry Entries for Servers	90
Keys Needed by DCOM	91
Keys Needed by Natural	92
Registry Entries for Clients	93

Using Statements and Commands in a NaturalX Server Environment	94
Using Statements and Commands in a NaturalX Server Environment	94
Natural Statements	94
DISPLAY, INPUT, PRINT, REINPUT and WRITE Statements	94
WRITE WORK FILE and READ WORK FILE Statements	95
STOP and TERMINATE Statements	95
Natural System Commands	95
NaturalX Glossary	96
NaturalX Glossary	96
Activation Policies	96
AppID	96
Class	96
Class GUID	96
Class Name	97
Class Module Name	97
COM	97
COM Class	97
DCOM	97
External Class	98
GUID	98
HFS	98
Instance	98
Interface	98
Interface GUID	99
Interface Inheritance	99
Internal Class	99
Local Class	99
Method	99
NaturalX Client	99
NaturalX Server	100
Natural Session	100
Object	100
Object Data Area - ODA	100
Object Data Variable	100
ProgID	100
Property	100
Registry	101
Registry Key	101
Server ID	101
Type Information	101
Type Library	101

NaturalX - Overview

The NaturalX documentation contains information required to administer and use NaturalX on all available platforms.

This documentation covers the following topics:

- Introduction to NaturalX
- Installing NaturalX
- Developing NaturalX Applications
- Distributing NaturalX Applications
- Security with NaturalX
- NaturalX Configuration Examples
- NaturalX System Registry Entries
- Using Statements and Commands in a NaturalX Server Environment
- NaturalX Glossary

For a detailed explanation of the symbols and operands used within the syntax descriptions, see the section Syntax Symbols and Operand Definition Table.

A document containing a number of frequently asked questions (FAQ) referring predominantly to mainframe issues with NaturalX is included in the Natural for Mainframes **Installation Guide**, see the section NaturalX Trouble Shooting FAQ, or click here if you have access to the mainframe online documentation.

Platform-Specific Information

Wherever necessary, platform-specific information in the present documentation is identified by the following terms:

Mainframe Refers to the operating systems OS/390, VSE/ESA, VM/CMS and BS2000/OSD, as well as all TP monitors supported by Natural under these operating systems.

OpenVMS When used in the present NaturalX documentation, the term refers only to the OpenVMS operating system running on Alpha/AXP systems.

UNIX Refers to all UNIX systems supported by Natural.

Windows Refers to the following operating systems:

In a Natural development environment:

- Microsoft Windows NT
- Microsoft Windows 2000

In a Natural run-time environment:

- Microsoft Windows 98
- Microsoft Windows NT
- Microsoft Windows 2000

OS/400 Refers to the OS/400 operating system running on AS/400 and iSeries 400 machines. See the documentation provided on the Natural for OS/400 product CD-ROM.

Introduction to NaturalX

This section covers the following topics:

- Why NaturalX?
 - Programming Techniques
-

Why NaturalX?

Software applications that are based on component architecture offer many advantages over traditional designs. These include the following:

- Faster development. Programmers can build applications faster by assembling software from prebuilt components.
- Reduced development costs. Having a common set of interfaces for programs means less work integrating the components into complete solutions.
- Improved flexibility. It is easier to customize software for different departments within a company by just changing some of the components that constitute the application.
- Reduced maintenance costs. In the case of an upgrade, it is often sufficient to change some of the components instead of having to modify the entire application.
- Easier distribution. Components encapsulate data structures and functionality in distributable units.

NaturalX enables you to create and distribute object-based applications. Using Distributed Object technology (currently DCOM), it enables you to:

- allow your components to be accessed by other components,
- execute these components on local and/or remote servers,
- access components written in a variety of programming languages across process and machine boundaries from within Natural programs,
- provide your existing Natural applications with (quasi) standardized interfaces.

The following scenario illustrates how a company could exploit these advantages. A company introduces a new sales management system that is based on an application design using components. There are numerous data entry components in the application, one for each sales point. But all of these sales point use a common tax calculation component that runs on a server. If the tax legislation is changed, then only the tax component has to be updated instead of changing the data entry components at each site. In addition, the life of the programmers is made easier because they do not have to worry about network programming, operating-system compatibility, and the integration of components that are written in different languages.

Programming Techniques

This section covers the following topics:

- Object-Based Programming
- Defining Classes
- Defining Interfaces
- Interface Inheritance

Object-Based Programming

NaturalX follows an object-based programming approach. Characteristic for this approach is the encapsulation of data structures with the corresponding functionality into classes. Encapsulation is a good basis for easy distribution. Because there are now (quasi) standards for the interoperation of software components on the basis of object models, an object-based approach is also a good basis for making software components interoperable across program, machine and programming language boundaries.

Defining Classes

In an object-based application, each function is considered to be a service that is provided by an object. Each object belongs to a class. Clients use the services either to perform a business task or to build even more complex services and to provide these to other clients. Hence the basic step in creating an application with NaturalX is to define the classes that form the application. In many cases, the classes simply correspond to the real things that the application in question deals with, for example, bank accounts, aircraft, shipments etc. There is a wide range of good literature about object-oriented design, and a number of well-proven methods can be used to identify the classes in a given business.

The process of defining a class can be broadly broken down into the following steps:

- Create a Natural module of type class.
- Specify the name of the class using the `DEFINE CLASS` statement. This name will be used by the clients to create objects of that class.
- Use the `OBJECT` clause of the `DEFINE DATA` statement to define how an object of the class will look internally. Create a local data area that describes the layout of the object with the data area editor, and assign this data area in the `OBJECT` clause.

These steps are described in more detail in the section *Developing Object-Based Natural Applications*.

Defining Interfaces

In order to be useful to clients, a class must provide services, which it does through interfaces. An interface is a collection of methods and properties. A method is a function that an object of the class can perform when requested by a client. A property is an attribute of an object that a client can retrieve or change. A client accesses the services by creating an object of the class and using the methods and properties of its interfaces.

The process of defining an interface can be broadly broken down into the following steps:

- Use the `INTERFACE` clause to specify an interface name.
- Define the properties of the interface with `PROPERTY` definitions.
- Define the methods of the interface with `METHOD` definitions.

These steps are described in more detail in the section *Developing Object-Based Natural Applications*.

Simple classes only have one interface, but a class may have more than one interface. This possibility can be used to group methods and properties into one interface that belong to the same functional aspect of the class and to define different interfaces to handle other functional aspects. For example, an *Employee* class could have an interface *Administration* that contains all of the methods and properties of the administrative aspects of an employee. This interface could contain the properties *Salary* and *Department* and the method *TransferToDepartment*. Another interface *Qualifications* could contain the qualification aspects of an employee.

Interface Inheritance

Defining several interfaces for a class is the first step towards using interface inheritance, which is a more advanced method of designing classes and interfaces. This makes it possible to reuse the same interface definition in different classes. Assume that there is a class *Manager*, which is to be treated in the same way as the class *Employee* with respect to qualification, but which is to be handled differently as far as administration is concerned. This can be achieved by having the *Qualification* interface in both classes. This has the advantage that a client that uses the *Qualification* interface on a given object does not have to check explicitly whether the object represents an *Employee* or a *Manager*. It can simply use the same methods and properties without having to know of what class the object is. The properties or methods can even be implemented in a different way in both classes provided they are presented through the same interface definition.

The process of using interface inheritance can be broadly broken down into the following steps:

- Use the INTERFACE statements to define one or more interfaces in a copycode instead of defining them directly in the class.
- The METHOD and PROPERTY definitions in the INTERFACE statement do not need to contain the IS clause. At this point, you just define the external appearance of the interface without assigning implementations to the methods and properties.
- Use the INTERFACE clause to include the copycode with its interface definition in each class that will implement the interface.
- Use the METHOD and PROPERTY statements to assign implementations to the methods and properties of the interface in each class that will implement the interface.

Installing NaturalX

This section describes how to install NaturalX and covers the following topics:

- Installing NaturalX under Windows 98/NT/2000
 - Installing NaturalX under UNIX
 - NaturalX System Architecture under OS/390
 - Installing NaturalX under OS/390
-

Installing NaturalX under Windows 98/NT/2000

NaturalX is part of Natural for Windows 98/NT/2000 and is automatically installed with the installation of Natural for Windows 98/NT/2000.

Installing NaturalX under UNIX

This section describes how to install NaturalX under UNIX.

Prerequisites

Before you begin to install NaturalX under UNIX, ensure that your computer meets the following prerequisites:

- EntireX DCOM Version 5.3.1
- For access from Windows NT to UNIX:
Windows NT Version 4.0 with Service Pack 6.
- Natural 5.1.1 for UNIX

Environment Variables

Make sure that the environment variables for EntireX DCOM have been set according to the EntireX DCOM documentation.

In order to start up correctly, Natural needs the environment variables NATDIR and NATVERS. To make sure that these environment variables are also set when a NaturalX server is launched automatically by EntireX DCOM, NATDIR and NATVERS must also be set in the environment where the ntd daemon (EntireX DCOM) is started. Set NATDIR to the path where Natural is installed and NATVERS to the name of the Natural version directory.

Installation Procedure

Step 1 - Perform the General Installation Procedure for Software AG Products for UNIX

For information on this subject, read Installing And Setting Up Software AG Products for UNIX in the Natural Installation and Operations Manual for OpenVMS and UNIX.

This section contains general information which applies when installing and setting up any Software AG product on a UNIX platform.

Step 2 - Execute the NaturalX Installation Script

To install NaturalX, you use the installation script "nxxinstall.bsh".

1. Issue the following commands to execute the installation script:

```
cd $NXXDIR/$NXXVERS/INSTALL.  
/nxxinstall.bsh
```

2. Follow the instructions provided by the installation script.
A new Natural nucleus with NaturalX support will be linked.
A backup copy of the file "natural" will be created and named "natural.old".

NaturalX System Architecture under OS/390

Before you start to install NaturalX under OS/390, it is important that you have a clear picture of the NaturalX system architecture under this operating system.

This section covers the following topics:

- Overview
- Environment Variables
- NaturalX Server Front-End
- NaturalX Output Files
- The NaturalX Server Monitor
- The DCOM Buffer Pool

Overview

NaturalX is available for OS/390 (Version 2.4 or above) and requires EntireX DCOM Version 5.2.1 or above. It uses the OS/390 UNIX Services and must have access to the OS/390 HFS (hierarchical file system). Usability is restricted to Natural sessions running in batch-oriented systems (TSO, batch and processes in OS/390 UNIX Services). NaturalX consists of the following four load modules:

- NATCOMST resides in a PDS load library. This module is the root entry point for the Natural COM function and is loaded by the Natural nucleus at session initialization. It provides the nucleus with a list of the entry points for the APIs in NATCOM.
- NATCOM resides in a PDS load library. This module contains the API functionality which enables the Natural nucleus to call functions in EntireX DCOM.
- The load module "naturalx" resides in the HFS and is used by EntireX DCOM to start a NaturalX DCOM server. NaturalX is a multi-threading application which is able to start and execute multiple Natural sessions on multiple clients simultaneously. The Natural sessions are executed within a configurable number of storage threads and the session data is swapped either using the Natural Roll Server or within the virtual storage. Swapping using virtual storage indeed limits the number of parallel sessions.
- The load module "natxmon" resides in the HFS and is used to monitor NaturalX server processes. For example, NATXMON can be used to terminate a specific NaturalX server process manually.

NaturalX Server

A NaturalX server is a process which is responsible for executing a Natural session which hosts classes, for further information, see the section Distributing Object-Based Natural Applications. Because mainframe Natural is a threading system, a NaturalX server on OS/390 is designed as a multi-user application able to maintain multiple Natural sessions for different clients. A class registered with activation policy *ExternalSingle* for example does not have to be hosted by an exclusive process, as it is hosted by an exclusive Natural session.

For sessions hosted by one server process, the following resources are exclusive:

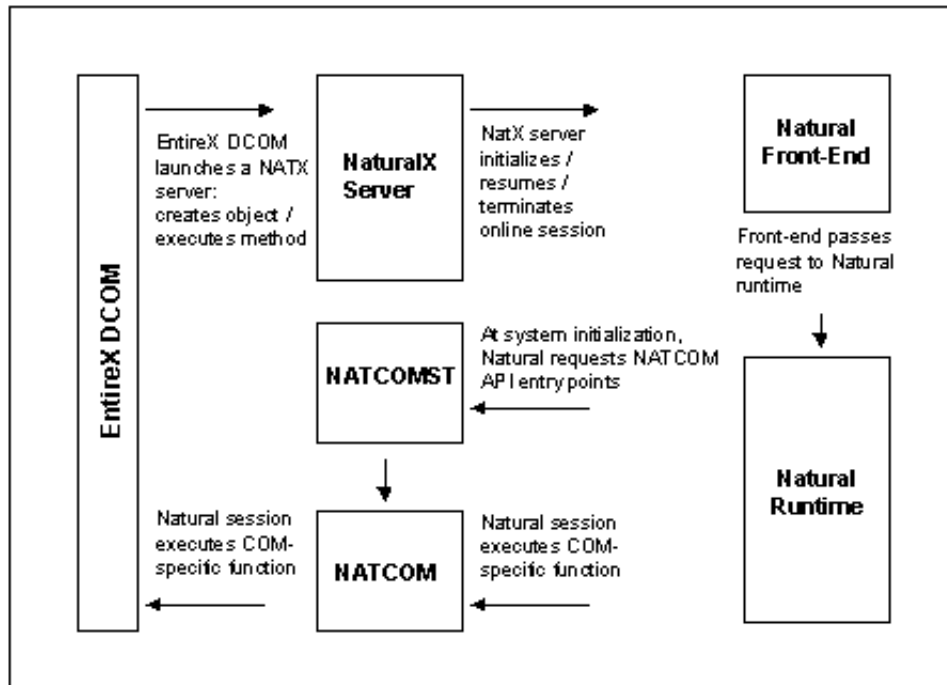
- Each session has its own database user ID,
- Each session has its own Natural session data (e.g. system variables, DATSIZE, error stack)

For sessions hosted by one server process, the following resources are common:

- All sessions run within one OS/390 address space.
- All sessions are started with the same session parameters.
- All DB2 access calls are made under the user ID of the server process.
- Print and work files can either be shared between sessions or be used exclusively by one session. For a detailed description, see the section **Natural as a Server** under **Natural in Batch Mode** in the section **Environment-Specific Information** in the Natural Operations for Mainframes documentation.

The following diagram gives an overview of the components involved in a NaturalX server environment:

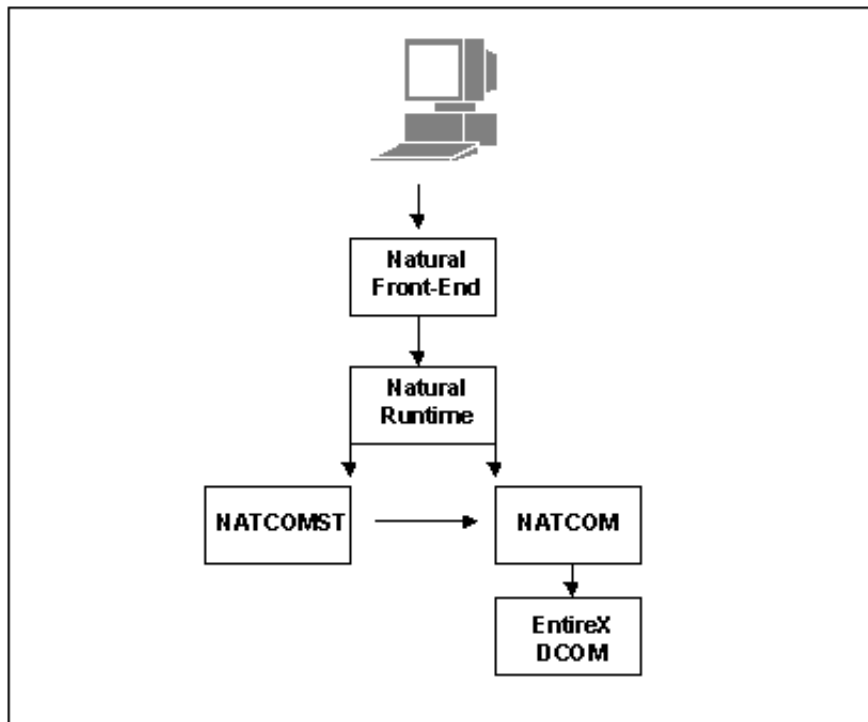
NaturalX Components in a Server Environment



NaturalX Client

A NaturalX COM client can be any Natural TSO or batch session which uses distributed COM objects. The following diagram gives an overview of the components involved in a NaturalX client session:

NaturalX Components in a Client Environment



Environment Variables

The execution of NaturalX is configured using UNIX environment variables. Some of these variables are required for COM clients, some for COM servers and some for both. The variables mentioned in the EntireX DCOM documentation must always be set. How the variables are set depends on how you start your Natural online session or your NaturalX COM server.

Starting Natural under TSO or Batch

If you start a Natural session in TSO or batch, an HFS file must be allocated to the DD name NATXENV. This HFS file contains the variable assignments.

Example of an HFS File Allocated to NATXENV

```
PATH=/u/dcomkit/SAG/dco/v411/bin:/u/nat/v311/bin
LIBPATH=/u/dcomkit/SAG/dco/v411/lib
_CEE_RUNOPTS=TERMTHDACT(MSG),POSIX(ON),STACK(4M,1M,ANY,KEEP),STORAGE(NONE,NONE,
NONE,8K),HEAP(32K,32K,ANYWHERE,KEEP,8K,4K)
SAGNODE=DAEY
NATDIR=/u/SAG/natural
NATVERS=v311
NATX_NUCNAME=NATOS31L
NATX_NTHREADS=2
NATX_THREADSIZE=300
NATX_TRACE=1
```

Example for TSO ISPF procedure

```
PROC 0
  CONTROL NOFLUSH ASIS LIST CONLIST
  ALLOC FILE(NATXENV) +
    PATH(' /u/SAG/NAT/V311/natxenv' ) PATHOPTS(ORDONLY)
  ALLOC FILE(CMPRINT) DA(*)
  ALLOC FILE(CMSYNIN) DA(*)
  ALLOC FILE(SYSOUT) DA(*)
  CALL 'SAG.NAT311.LOAD(NAT31)'
```

Starting NaturalX Servers Manually

Usually EntireX DCOM launches the NaturalX server automatically if any client requests a server which is currently not active. It is always possible to start a NaturalX COM server manually under the OS/390 UNIX shell by executing the executable "naturalx". The variables must be defined in the shell environment from which you start NaturalX. NaturalX requires the session parameter COMSERVERID=*serverid* (for OS/390, DCOM=(SERVID=*serverid*)).

Example

```
naturalx "DCOM=(SERVID=MYSERVER)"
```

It is not possible to start a conventional Natural session as a COM server. A NaturalX COM server must be started by the executable "naturalx".

Starting NaturalX Servers from EntireX DCOM

If EntireX DCOM (RPCSS) launches the NaturalX COM server, NaturalX inherits the environment variables from EntireX DCOM. All the environment variables required by NaturalX must be defined in the shell environment from which EntireX DCOM is started.

List of NaturalX Environment Variables

In the sections below, the characters in brackets indicate whether a variable is required for clients (C), servers (S) or both (C/S):

DCOLIB - C

Points to the directory in which the standard OLE type library "stdole32.tlb" is stored. "stdole32.tlb" is included in the type library which is created when a class is registered.

Example for EntireX DCOM 5.2.2

```
DCOLIB= "/EXXDIR/EXXVERS/lib"
```

NATDIR - C/S

Points to the Natural HFS root directory.

Example

```
NATDIR=/u/SAG/natural/
```

NATVERS - C/S

Specifies the version-dependent subdirectory of NATDIR.

Example

```
NATVERS=v311
```

NATX_DELAY

Usually a COM server terminates if the number of objects hosted is zero. NATX_DELAY is used to delay server termination.

Examples

```
NATX_DELAY=30S causes termination delay of 30 seconds
NATX_DELAY=10M causes termination delay of 10 minutes
NATX_DELAY=1H causes termination delay of 1 hour
NATX_DELAY=INFINITE causes no termination
```

NATX_DYNALLOC

$$\text{NATX_DYNAL LOC} = \left\{ dd\text{-}name, \left\{ \begin{array}{l} \text{HFS} = \text{hfs-file name} \\ \text{PDS} = \text{dataset-name} \\ \text{SYSOUT} = \text{output-class} \end{array} \right\} \right\} : \dots$$

Specifies DD name allocations for NaturalX servers by defining a list of DD names which are allocated to datasets.

dd-name	DD name with max. 8 characters (for example SYSUDUMP)
hfs-filename	hfs file name with absolute path definition
dataset-name	existing and catalogued dataset name
output-class	single character JES output class

Example

```
NATX_DYNALLOC="CMPRINT,HFS=/u/tmp/myfile:SYSUDUMP,PDS=NAT.DUMP.F01:CMPT01,SYSOUT=X"
```

NATX_DYNALLOC allocates the DD name

- CMPRINT to HFS file 'myfile' on directory /u/tmp
- SYSUDUMP to dataset NAT.DUMP.F01
- CMPT01 to output class X.

NATX_FEOPT - S

Specifies additional options for the Natural front-end as follows:

01	Do not use the roll server
02	Clean up roll file at server termination

Example

```
NATX_FEOPT=01
```

NATX_FEPRM

Specifies additional Natural Front-End parameters as specified in the Startup Parameter Area. You can define multiple parameters. Each parameter is specified by a pair of 8-character strings of which the first contains the parameter keyword and the second, the parameter value. For further information, see the Natural Operations for Mainframe documentation, section **Environment-Specific Information**, section **Natural in Batch Mode**.

Example

```
NATX_FEPRM="MSGCLASSX"
```

This setting determines that the default output class for CMPRINT is "X".

NATX_INITTOUT

Specifies the number of seconds the client must wait until the launched NaturalX server is initialized. The default value is 10.

Example

```
NATX_INITTOUT=5
```

The client waits at most 5 seconds until the server is initialized.

The error message NAT0711 with the DCOM code 8004100C occurs if the timeout limit is reached.

NATX_NTHREADS - S

The number of physical storage threads to be allocated by the Natural front-end. The number of sessions which can be executed in parallel.

Note:

This number does not limit the number of sessions within the server, but the number of sessions which can be in execution status concurrently. The number of sessions is limited by the size of the Natural swap medium.

Example

```
NATX_NTHREADS=5
```

NATX_NUCNAME - S

The name of the Natural front-end to start a Natural session. The front-end resides on a PDS member. For further information, see the section NaturalX Server Frontend.

Example

```
NATX_NUCNAME=NAT311SV
```

NATX_THREADSIZE - S

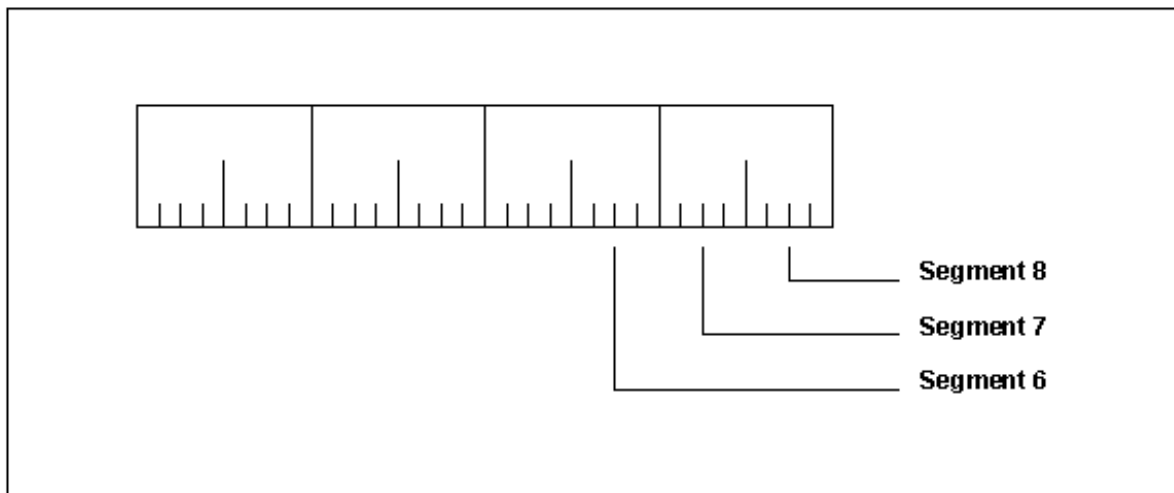
The size (in KB) of each physical storage thread which contains the Natural session data at execution time.

Example

```
NATX_THREADSIZE=800
```

NATX_TRACE - S

Defines the trace level of the server. The value is used as an 8 X 4-bit flag container in which each 4-bit segment controls the trace of a particular NaturalX functional unit.



Segment	Description
1 - 5	Not used.
6	Controls the trace output of NATCOM client calls.
7	Controls the trace output of server calls
8	Controls the trace output of the NaturalX front-end stub.

A segment consists of four bits where the first bit is currently not used. The value of a segment can thus be in the range 0 - 7. The higher the value, the more extensive the trace output.

Example

```
NATX_TRACE=0x00000172
```

This setting determines a level 1 trace for client calls, a level 7 trace for server calls and a level 2 trace for the NaturalX font-end stub.

PATH - C/S

Refers to the DCOM and Natural bin directory. For further information, see your OS/390 documentation.

Example

```
PATH=/u/SAG/dco/bin:u/SAG/nat/bin
```

STEPLIB - S

Refers to the PDS where the Natural front-end is installed. For further information, see your OS/390 documentation.

Example

```
STEPLIB="RZ.NAT311.LOAD"
```

Note:

The PDS load libraries defined in the STEPLIB environment variable must be defined in the 'system sanction list for set-user-id and set-group-id programs'. The 'system sanction list' is a HFS file containing the dataset names available for processes which are invoked under a different user-ID. For more information on the 'system sanction list', see the description of the statement STEPLIBLIST in the IBM manual OS/390 V2R4.0 OpenEdition Planning.

NaturalX Server Front-End

The NaturalX server front-end is a standard MVS batch driver which is assembled with the NTOS parameter LE370=POSIX. The same applies to the TSO driver if you execute COM requests within your TSO Natural online session.

NaturalX Output Files

For tracing purposes NaturalX allocates three output files for each COM server ID and three output files for each client user ID. The files can be found under:

```
NATDIR/NATVERS/trace/server/serverid.trc  the stub trace file
                                /serverid.sto  the stub stdout file
                                /serverid.ste  the stub stderr file
```

and

```
NATDIR/NATVERS/trace/client/userid.trc    the stub trace file
                                /userid.sto  the stub stdout file
                                /userid.ste  the stub stderr file
```

The stdout and stderr files for clients do not exist if the client session is hosted by a COM server (that is, a COM server session becomes an agent). In that case the output of stderr and stdout is directed to the appropriate server file. The output files are always removed at server/client initialization.

The NaturalX Server Monitor

The server monitor "natxmon" can be used to send messages to a specific NaturalX server process via standard UNIX message queues in order to administer a server process without using COM. "natxmon" is used from the OS/390 UNIX shell with the following two parameters:

natxmon pid message

Parameter	Description
<i>pid</i>	Contains the process ID of the server process.
<i>message</i>	Contains one of the following numeric values: <ol style="list-style-type: none"> 1. The server writes status information to the trace file. 2. The server deregisters and terminates. 3. The server aborts.

Example

```
natxmon 4711 2
```

This setting terminates the NaturalX server with the PID 4711.

The DCOM Buffer Pool

The COM implementation is based on VTABLES (virtual function tables) which contain lists of entry points which are used to call the object methods. A VTABLE is not relocatable because its address itself identifies a specific class object. Natural collates all VTABLES within a DCOM buffer pool. The allocation is similar to already existing Natural buffer pools, for example, the Sort and Editor buffer pools. A new buffer pool type DCOM has been introduced. For further information, see the BPI parameter documentation. The buffer pool type DCOM can be either local or global.

Installing NaturalX under OS/390

Installing NaturalX under OS/390 is described in the Natural Installation Guide for Mainframes.

Developing NaturalX Applications

The first step in creating a Natural application that utilizes the advantages of distributed object computing is to develop the code, which is then distributed in the second step. This section tells you how to develop an application by defining and using classes.

This section covers the following topics:

- Using the Class Builder
 - Defining Classes
 - Using Classes and Objects
 - *THIS-OBJECT System Variable
-

Using the Class Builder

On Windows NT and Windows 2000, Natural provides the Class Builder as the tool to develop Natural classes. The Class Builder shows a Natural class in a structured hierarchical order and allows the user to manage the class and its components efficiently. If you use the Class Builder, no knowledge or only a basic knowledge of the syntax elements described in the section Defining Classes is required. On other platforms, you develop classes using the Natural Program Editor. In this case, you should know the syntax of class definition described in the section Defining Classes.

Defining Classes

When you define a class, you must create a Natural class module, within which you create a `DEFINE CLASS` statement. Using the `DEFINE CLASS` statement, you assign the class an externally usable name and define its interfaces, methods and properties. You can also assign an object data area to the class, which describes the layout of an instance of the class. The `DEFINE CLASS` statement is also used to supply a global unique identifier to those classes that are to be registered with DCOM.

This section covers the following topics:

- Creating a Natural Class Module
- Specifying a Class
- Defining an Interface
- Assigning an Object Data Variable to a Property
- Assigning a Subprogram to a Method
- Implementing Methods

Creating a Natural Class Module

 **To create a Natural class module**

- Create a Natural object of type Class.

Specifying a Class

The `DEFINE CLASS` statement defines the name of the class, the interfaces the class supports and the structure of its objects. For classes that are to be registered with DCOM, it specifies also the Globally Unique ID of the class and its Activation Policy.

 **To specify a class**

- Use the DEFINE CLASS statement as described in the Natural Statements documentation.

Defining an Interface

Each interface of a class is specified with an INTERFACE statement inside the class definition. An INTERFACE statement specifies the name of the interface and a number of properties and methods. For classes that are to be registered with DCOM, it specifies also the Globally Unique ID of the interface.

A class can have one or several interfaces. For each interface, one INTERFACE statement is coded in the class definition. Each INTERFACE statement contains one or several PROPERTY and METHOD clauses. Usually the properties and methods contained in one interface are related from either a technical or a business point of view.

The PROPERTY clause defines the name of a property and assigns a variable from the object data area to the property. This variable is used to store the value of the property.

The METHOD clause defines the name of a method and assigns a subprogram to the method. This subprogram is used to implement the method.

 **To define an interface**

- Use the INTERFACE statement as described in the Natural Statements documentation.

Assigning an Object Data Variable to a Property

The PROPERTY statement is used only when several classes are to implement the same interface in different ways. In this case, the classes share the same interface definition and include it from a Natural Copycode. The PROPERTY statement is then used to assign a variable from the object data area to a property, outside the interface definition. Like the PROPERTY clause, the PROPERTY statement defines the name of a property and assigns a variable from the object data area to the property. This variable is used to store the value of the property.

 **To assign an object data variable to a property**

- Use the PROPERTY statement as described in the Natural Statements documentation.

Assigning a Subprogram to a Method

The METHOD statement is used only when several classes are to implement the same interface in different ways. In this case, the classes share the same interface definition and include it from a Natural Copycode. The METHOD statement is then used to assign a subprogram to the method, outside the interface definition. Like the METHOD clause, the METHOD statement defines the name of a method and assigns a subprogram to the method. This subprogram is used to implement the method.

 **To assign a subprogram to a method**

- Use the METHOD statement as described in the Natural Statements documentation.

Implementing Methods

A method is implemented as a Natural subprogram in the following general form:

```
DEFINE DATA statement  
*  
* Implementation code of the method  
*  
END
```

For information on the DEFINE DATA statement see the Natural Statements Manual.

All clauses of the DEFINE DATA statement are optional.

It is recommended that you use data areas instead of inline data definitions to ensure data consistency.

If a PARAMETER clause is specified, the method can have parameters and/or a return value.

Parameters that are marked 'BY VALUE' in the parameter data area are input parameters of the method.

Parameters that are not marked 'BY VALUE' are passed *by reference* and are input/output parameters. This is the default.

The first parameter that is marked 'BY VALUE RESULT' is returned as the return value for the method. If more than one parameter is marked in this way, the others will be treated as input/output parameters.

Parameters that are marked 'OPTIONAL' are available with Version 4.1.2 and all subsequent releases. Optional parameters need not to be specified when the method is called. They can be left unspecified by using the nX notation in the SEND METHOD statement.

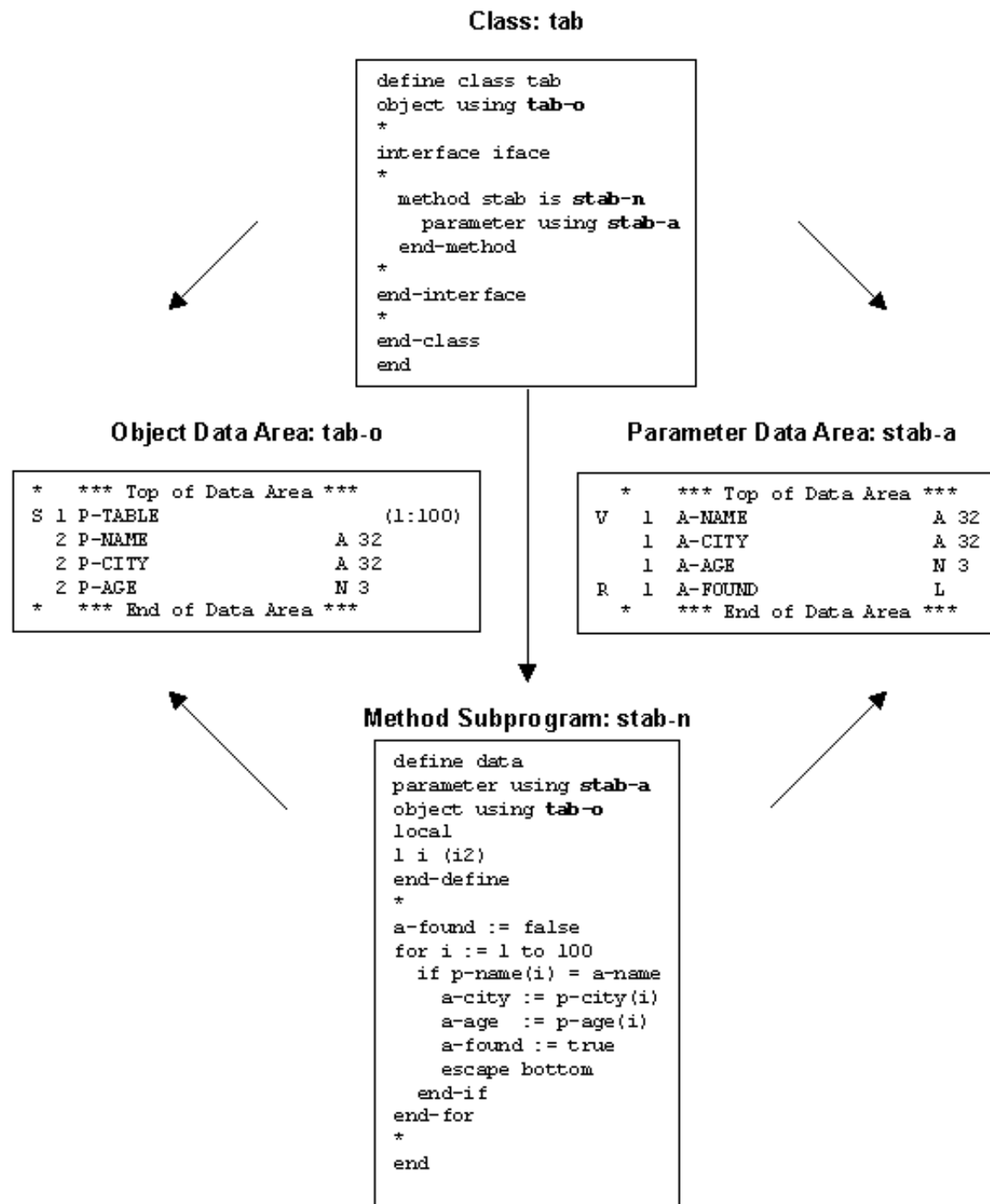
To make sure that the method subprogram accepts exactly the same parameters as specified in the corresponding METHOD statement in the class definition, use a parameter data area instead of inline data definitions. Use the same parameter data area as in the corresponding METHOD statement.

To give the method subprogram access to the object data structure, the OBJECT clause can be specified. To make sure that the method subprogram can access the object data correctly, use a local data area instead of inline data definitions. Use the same local data area as specified in the OBJECT clause of the DEFINE CLASS statement.

The GLOBAL, LOCAL and INDEPENDENT clauses can be used as in any other Natural program.

While technically possible, it is usually not meaningful to use a CONTEXT clause in a method subprogram.

The following example retrieves data about a given person from a table. The search key is passed as a 'BY VALUE' parameter. The resulting data is returned through 'BY REFERENCE' parameters ('BY REFERENCE' is the default definition). The return value of the method is defined by the specification 'BY VALUE RESULT'.



Using Classes and Objects

Objects created in a local Natural session can be accessed directly and objects created in other processes or on remote machines can be accessed via DCOM. In both cases the rules for accessing and using classes and their objects are the same. The statement `CREATE OBJECT` is used to create an object (also known as an instance) of a given class. To reference objects in Natural programs, object handles have to be defined in the `DEFINE DATA` statement. Methods of an object are invoked with the statement `SEND METHOD`. Objects can have properties, which can be accessed using the normal assignment syntax.

Note:

Classes created with NaturalX can all be used by COM, once they have been registered.

This section covers the following topics:

- Defining Object Handles
- Creating an Instance of a Class
- Invoking a Particular Method of an Object
- Accessing Properties
- Sample Application

Defining Object Handles

To reference objects in Natural programs, object handles have to be defined as follows in the DEFINE DATA statement:

```

DEFINE DATA ...
    level handle-name [(array-definition)] HANDLE OF OBJECT
    ...
END-DEFINE

```

Example

```

DEFINE DATA LOCAL
    1 #MYOBJ1 HANDLE OF OBJECT
    1 #MYOBJ2 (1:5) HANDLE OF OBJECT
END-DEFINE

```

Creating an Instance of a Class

► To create an instance of a class

- Use the CREATE OBJECT statement as described in the Natural Statements documentation.

Invoking a Particular Method of an Object

► To invoke a particular method of an object

- Use the SEND METHOD statement as described in the Natural Statements documentation.

Accessing Properties

Properties can be accessed using the ASSIGN (or COMPUTE) statement as follows:

```

ASSIGN operand1.property-name = operand2
ASSIGN operand2 = operand1.property-name

```

Object Handle - operand1

Operand1 must be defined as an object handle and identifies the object whose property is to be accessed. The object must already exist.

operand2

As *operand2*, you specify an operand whose format must be data transfer-compatible to the format of the property. Please refer to the data transfer compatibility rules in the Natural Reference documentation for further information.

If the object is to be accessed via DCOM, you must also take into account the rules for data type conversion which are outlined in the section Data Type Conversions.

property-name

The name of a property of the object.

If the property name conforms to Natural identifier syntax, it can be specified as follows

```
create object #o1 of class "Employee"
  #age := #o1.Age
```

If the property name does not conform to Natural identifier syntax, it must be enclosed in angle brackets:

```
create object #o1 of class "Employee"
  #salary := #o1.<<%Salary>>
```

The property name can also be qualified with an interface name. This is necessary if the object has more than one interface containing a property with the same name. In this case, the qualified property name must be enclosed in angle brackets:

```
create object #o1 of class "Employee"
  #age := #o1.<<PersonalData.Age>>
```

Example

```
define data
  local
    1 #i          (i2)
    1 #o          handle of object
    1 #p          (5) handle of object
    1 #q          (5) handle of object
    1 #salary     (p7.2)
    1 #history    (p7.2/1:10)
  end-define
  * ...
  * Code omitted for brevity.
  * ...
  * Set/Read the Salary property of the object #o.
  #o.Salary := #salary
  #salary := #o.Salary
  * Set/Read the Salary property of
  * the second object of the array #p.
  #p.Salary(2) := #salary
  #salary := #p.Salary(2)
  *
  * Set/Read the SalaryHistory property of the object #o.
  #o.SalaryHistory := #history(1:10)
  #history(1:10) := #o.SalaryHistory
  * Set/Read the SalaryHistory property of
```

```

* the second object of the array #p.
#p.SalaryHistory(2) := #history(1:10)
#history(1:10) := #p.SalaryHistory(2)
*
* Set the Salary property of each object in #p to the same value.
#p.Salary(*) := #salary
* Set the SalaryHistory property of each object in #p
* to the same value.
#p.SalaryHistory(*) := #history(1:10)
*
* Set the Salary property of each object in #p to the value
* of the Salary property of the corresponding object in #q.
#p.Salary(*) := #q.Salary(*)
* Set the SalaryHistory property of each object in #p to the value
* of the SalaryHistory property of the corresponding object in #q.
#p.SalaryHistory(*) := #q.SalaryHistory(*)
*
end

```

In order to use arrays of object handles and properties that have arrays as values correctly, it is important to know the following:

A property of an occurrence of an array of object handles is addressed with the following index notation:

```
#p.Salary(2) := #salary
```

A property that has an array as value is always accessed as a whole. Therefore no index notation is necessary with the property name:

```
#o.SalaryHistory := #history(1:10)
```

A property of an occurrence of an array of object handles which has an array as value is therefore addressed as follows:

```
#p.SalaryHistory(2) := #history(1:10)
```

Sample Application

An example application is provided in the libraries SYSEXCOM and SYSEXCOC. See the A-README members in these libraries for information about how to run the example.

***THIS-OBJECT System Variable**

Format/length: HANDLE OF OBJECT.

Content modifiable: No.

This system variable is a handle to the currently active object. The currently active object uses the *THIS-OBJECT system variable either to execute its own methods or to pass a reference to itself to another object.

*THIS-OBJECT only contains an actual value when a method is being executed. Otherwise it contains NULL-HANDLE.

The following statements may occur in a method implementation:

Example

```
define data
  ...
  local
  1 #self handle of object
  1 #another handle of object
end-define
...
* Calling the current object's own methods.
* ("Hello" is a method of the current object.)
send "Hello" to *this-object
...
* Passing a handle to the current object
* in a method call to another object.
#self := *this-object
send "ItsMe" to #another with #self
...
*
end
```

Distributing NaturalX Applications

On Windows, UNIX and OS/390 platforms, NaturalX applications can be distributed using DCOM. Most information in this section is of a general nature. The section Configuration Examples supplies platform-specific information.

This section covers the following topics:

- General
 - Globally Unique Identifiers (GUIDs)
 - NaturalX Servers
 - Activation Policies
 - Registration
 - DCOMPARM System Command (OS/390 Only)
 - Type Information
 - Configuration Overview
 - Sample Application
-

General

Using NaturalX, you can make Natural classes and their services available to local and remote clients, thus creating distributed applications. Local clients are processes that run on the same machine as a given NaturalX server, and remote clients are processes that run on a different machine.

In order to distribute applications, a widely-used distributed object technology is used - the Microsoft Distributed Component Object Model (DCOM). When you register a Natural class to DCOM, its interfaces are presented to clients in a quasi-standardized fashion as dynamic COM interfaces, which are also known as dispatch interfaces. These interfaces can be easily addressed by many programming languages including Visual Basic, Java, C++ and, of course, Natural.

There are several points that must be taken into consideration when organizing the distribution of a NaturalX application. Each of these points is discussed in more detail in this chapter.

- Determine whether each class should be internal, external or local (see the section Internal, External and Local Classes).
- Globally unique IDs (GUIDs) must be assigned to the internal and external classes and their interfaces in order to be able to address them uniquely in the network (see the section Globally Unique Identifiers (GUIDs)).
- You can define the activation policy for each class in order to control the conditions under which DCOM activates it (see section Activation Policies).
- In order to organize classes to applications, you can define NaturalX servers and assign the classes to them (see the section NaturalX Servers).
- Classes must be registered to make them known to DCOM (see section Registration).
- You can configure an application in order to further control its behavior (see the sections Configuration Overview and Configuration Examples).

Internal, External and Local Classes

It is important to distinguish between classes for internal use, classes for external use and those for local use only.

Internal Classes

The most important feature of internal classes is that their objects (instances) can only be created in the local client process.

Internal classes have the following features:

- Access to client session-dependent resources such as files and system variables.
- Can run within the client transaction.
- Can be debugged using the Natural Debugger.

External Classes

Windows 98/NT/2000 and UNIX

An external class can be created in the client process provided that the client process is simultaneously a server for the class. In addition, an external class can be debugged with the Natural Debugger (remote debugging).

OS/390

The most important feature of external classes is that their objects (instances) can *not* be created in the local client process.

All Platforms

External classes have the following features on all platforms:

- No access to client session-dependent resources such as stacks, files and system variables.
- Do not run within the client transaction.
- Can be used by remote nodes.
- Can be used by various clients using a variety of languages such as Natural, Java, Visual Basic, C/C++, etc.

Local Classes

Local classes are classes which are executed in local execution mode. Natural executes a class locally (within the Natural session) if it is either not registered or if DCOM is not available.

Local classes have the following features:

- Can be used even if DCOM is not available.
- Need not be registered with DCOM.
- Cannot be used from outside the client process.

Globally Unique Identifiers - GUIDs

DCOM uses global unique identifiers (GUIDs) - 128-bit integers that are virtually guaranteed to be unique throughout the world - to identify every interface and every class. This helps to ensure that server components can be located and to prevent clients connecting to an object accidentally.

If a class is to be registered to DCOM, every interface defined in a Natural class and the class itself must be associated with such a globally unique ID.

Once a globally unique ID has been assigned to an interface or a class, the ID must never be changed.

Using the Class Builder

On Windows NT and Windows 2000, Natural provides the Class Builder as the tool to develop Natural classes. The Class Builder automatically assigns a GUID to every class and interface.

Using the Data Area Editor

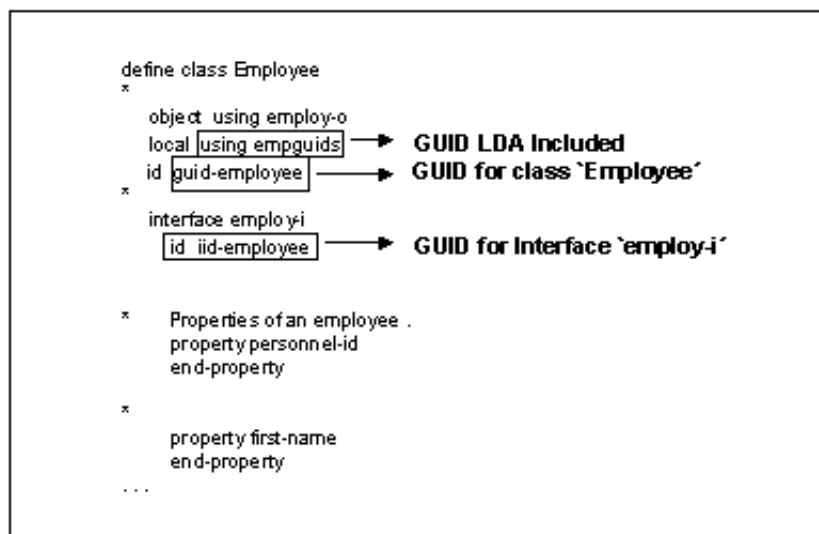
On OS/390 and UNIX, you must use the Data Area Editor to create GUIDs. GUIDs are alphanumeric constants of type A36 in Natural. If you use this approach, it is suggested that, for ease of administration, you use one Local Data Area to store all the GUIDs for one project or application.

Note:

GUID generation is a functionality of COM and on the mainframe available in Batch and TSO only.

1. Create an LDA and insert one or more GUIDs, each provided with a symbolic name.
For further information, see your Natural User's Guide.
These symbolically-named constants are inserted into the data area and are initialized with an internally-generated globally unique ID.
2. Include the GUID LDA at the top of the class and use these named constants in your class and interface definition.

Example Including an LDA containing a GUID Definition in a Class Definition



NaturalX Servers

This section covers the following topics:

- COM Classes and Servers
- NaturalX Classes and Servers
- NaturalX Servers and Natural Sessions under OS/390
- NaturalX Servers and Natural Sessions under Windows 98/NT/2000 and UNIX
- The Role of the Server ID
- Organizing Server IDs

COM Classes and Servers

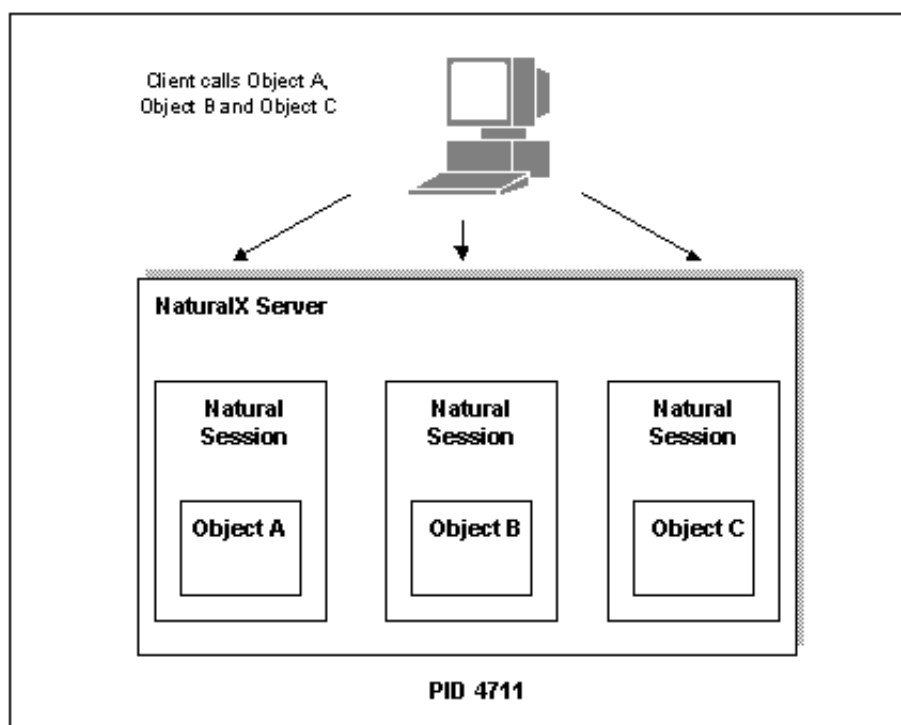
Each COM class must be hosted by a server process. The server process has a number of administrative and technical responsibilities, such as making the class and its interfaces available to DCOM and maintaining the memory occupied by the objects created. Whenever a client requests a new object of a certain class, DCOM checks whether the corresponding server process is already running. If this is not the case, DCOM launches it and passes the request to the server. When the server starts up, it makes its classes available to DCOM. While the server is running, it executes client requests for creation and deletion of objects and execution of methods. When the last object maintained by a server is deleted, the server shuts down automatically. For more detailed information about DCOM classes and servers, please refer to the Microsoft DCOM specification.

NaturalX Classes and Servers

Classes implemented with Natural can be made accessible as DCOM classes. But with Natural, it is not necessary to implement DCOM servers to host the classes. Instead, NaturalX itself performs the tasks of a DCOM server. NaturalX acts as a generic DCOM server for all classes written in Natural. The task that remains for a Natural class developer is just to implement the classes and to assign them to a NaturalX server.

NaturalX Servers and Natural Sessions under OS/390

Under OS/390, a Natural DCOM server process can manage several Natural sessions in parallel. This means that objects from different clients which request the same server are all hosted by the same server process (region). Each client exclusively owns its own Natural session.



Starting NaturalX Servers

EntireX DCOM launches a NaturalX server automatically if a client requests a server which is not currently active. You can also start a NaturalX server manually under OS/390 UNIX services, under TSO or in batch by executing the load module "naturalx" as follows:

Example for OS/390 UNIX Shell

```
naturalx "fuser=(10,32) profile=(dcomqa, 10,930) DCOM=(SERVID=gatest01)"
```

Example for TSO

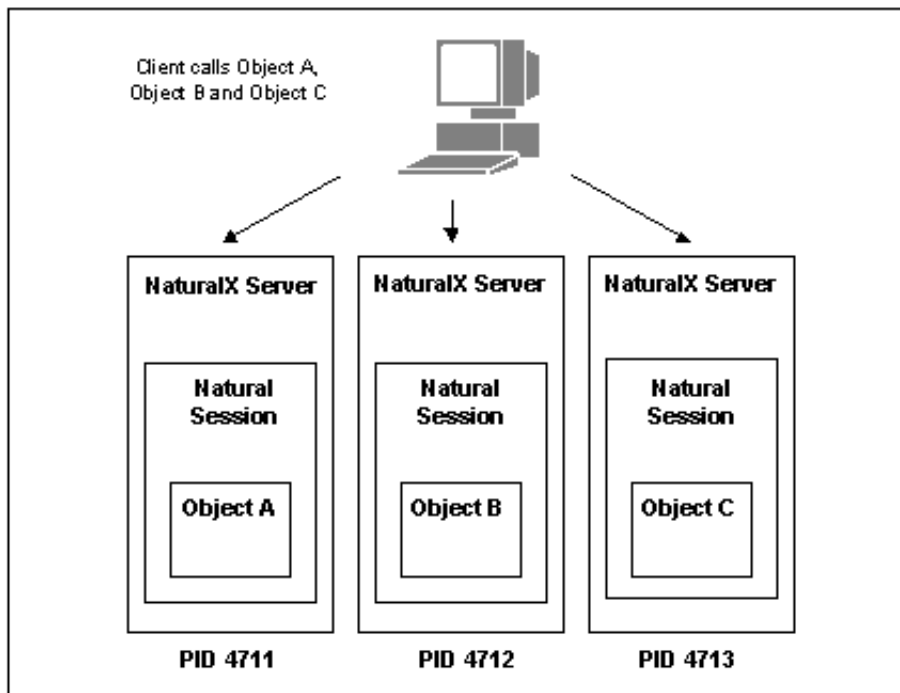
```
PROC 0
  CONTROL NOFLUSH ASIS LIST CONLIST
  ALLOC FILE(NATXENV) +
    PATH('/u/nat/natxenv') PATHOPTS(ORDONLY)
  ALLOC FILE(CMPRINT) DA(*)
  ALLOC FILE(CMSYNIN) DA(*)
  ALLOC FILE(SYSOUT) DA(*)
  ISPEXEC LIBDEF ISPLLIB DATASET ID('PRD.NXX111.LOAD' -
    'PRD.NAT311.LOAD')
  CALL 'PRD.NXX111.LOAD(NATURALX)' +
    'profile=(dcomqa,10,930),dcom=(servid=gatest01)'
END
```

Example for Batch

```
//NXSVR JOB CLASS=K,MSGCLASS=X
//NX      EXEC PGM=NATURALX,REGION=3200K,
// PARM=('profile=(dcomqa,10,930)',,
// 'dcom=(servid=qatest01)')
//STEPLIB DD DISP=SHR,DSN=PRD.NXX111.LOAD
//        DD DISP=SHR,DSN=PRD.NAT311.LOAD
//SYSUDUMP DD SYSOUT=X
//NATXENV DD PATH='/u/nat/natxenv',PATHOPTS=(ORDONLY)
//CMPRINT DD SYSOUT=X
/*
```

NaturalX Servers and Natural Sessions under Windows 98/NT/2000 and UNIX

Under Windows 98/NT/2000 and UNIX, each Natural session runs in its own exclusive NaturalX server process.



The Role of the Server ID

One of the tasks of a DCOM server is to make its classes available to DCOM during startup. But since NaturalX acts as a generic DCOM server, it has no built-in knowledge about the classes it shall provide. Instead, it finds the list of these classes in the system registry under the key of its server ID. The server ID is a Natural-owned key in the system registry, keeping together all classes that belong to a given NaturalX server. It is an arbitrary alphanumeric string of 32 characters which does not contain blanks and which is not case sensitive.

How does a NaturalX server know under which server ID it is running? The server ID is defined with the Natural parameter `COMSERVERID=servid` (for OS/390, `DCOM=(SERVID=servid)`). This parameter is either passed to a NaturalX server as a dynamic parameter on the command line, or it is defined in the Natural parameter module.

How are classes assigned to server IDs? Assume Natural has been started with a certain server ID. Then every class that a user registers during this Natural session is entered into the system registry under the current server ID.

Server IDs provide a means of grouping classes created in Natural and assigning them to different NaturalX server processes. The use of server IDs is, however, not compulsory: if Natural is started without a server ID, all Natural classes are registered under the predefined server ID "Default".

Example

Consider the example *Employees* application consisting of the classes *DepartmentList*, *EmployeesList* and *Employee* (this application is contained in the example library SYSEXCOM). These three classes are to be hosted by a NaturalX server called *Employees*.

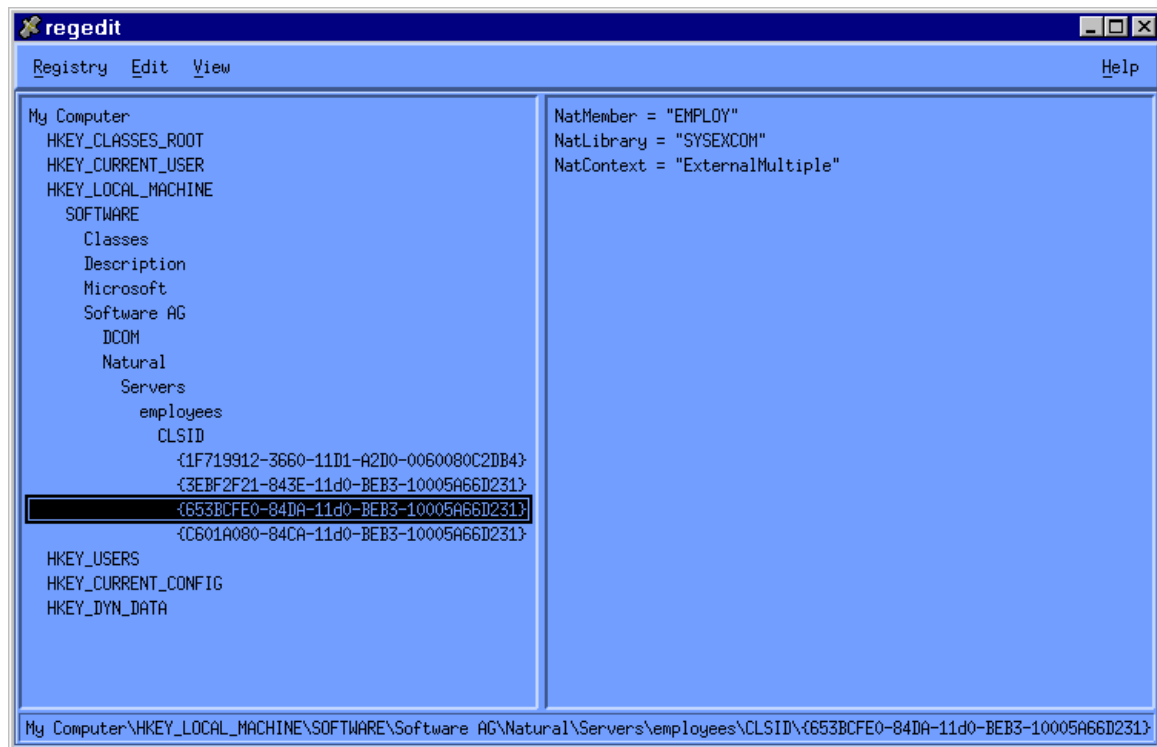
1. Start Natural with the desired server ID.
2. Logon to the library SYSEXCOM.
LOGON SYSEXCOM
3. Register the classes with the REGISTER command on the Natural command line
REGISTER *

Note:

On the mainframe, the REGISTER command is only available under TSO and in batch.

For further information, see the section The REGISTER Command.

The three classes are now registered under the server ID *Employees*. The following example shows the Registry Editor under UNIX.



Whenever an object of one of these classes is requested, DCOM will start a NaturalX server process with the server ID *Employees*, which will then provide the classes.

Organizing Server IDs

The server ID represents the set of all classes that are made available to DCOM when the corresponding NaturalX server is started. It is recommended that you group under one server ID those classes that form an application from the business point of view, or that otherwise belong together logically. Similarly, classes that are never used in the same context should be registered under different server IDs. Another criterion for the assignment of classes to server IDs is security (see the section Security). From this aspect, it makes sense to group under the same server ID those classes for which common authorizations will be defined.

Activation Policies

This section covers the following topics:

- Activation Policies Under Windows 98/NT/2000 and UNIX
- Activation Policies Under OS/390
- Setting Activation Policies
- When to use which Activation Policy

Activation Policies Under Windows 98/NT/2000 and UNIX

If a client makes a request to create an object of a certain class, it is DCOM's task to start a server process that provides the class and to direct the request to this process. For Natural classes, the responsible server process is a NaturalX server. DCOM recognizes different options that control when a new server process is started or when an object is created in a server process that is already running. For further information, see the section Registration. While registering a Natural class with the REGISTER command, you can control which activation options DCOM shall use for this class. NaturalX combines the different options supported by DCOM in the form of the following three activation policies:

- *ExternalMultiple*
If a Natural class is registered with the activation policy *ExternalMultiple*, and a client requests an object of that class, DCOM tries first to create the requested object in the current process. Remember that the client itself might at the same time be a NaturalX server and might provide the class itself. If the current process is not a server for the class, DCOM starts a new NaturalX server process and creates the object in that process. If a second object of the same class is created later, this object is also created in that server process. This means that the same server process can contain several objects of the class.
- *ExternalSingle*
If a Natural class is registered with the activation policy *ExternalSingle*, DCOM starts a new NaturalX server process each time an object of this class is created. One server process can contain only one object of the class.
- *InternalMultiple*
If a Natural class is registered with the activation policy *InternalMultiple*, DCOM always creates objects of this class in the current process. The same server process can contain several objects of the class.

The default activation policy is *ExternalMultiple*. This default is defined with the Natural parameter ACTPOLICY and can be changed with the NATPARM utility.

Activation Policies Under OS/390

Activation policies are used to decide whether a class requires an exclusive Natural session or whether it can share one with other objects. As far as DCOM is concerned, a NaturalX server is responsible for all the objects of those classes which are registered with the same server ID. That is, for any one server ID, only one NaturalX server process is active.

Activation policy is evaluated by the NaturalX server. The following two sets of options are supported:

- **External/Internal**
Determines whether an object is created in the Natural client session requesting it (internal), or in a different Natural session (external).
- **Single/Multiple**
Determines whether an object requires a Natural session for its exclusive use (single) or not (multiple).

NaturalX combines the above options to form the following three activation policies with which classes can be registered:

- *InternalMultiple*
An object is created within the current Natural session.
- *ExternalMultiple*
An object is created in a different Natural session which may accomodate multiple objects. Because it is not possible to distinguish between multiple client applications running under the same user ID on the same node, all objects created by the same user on the same node are hosted by the same single Natural session. This is valid even if the objects are created from within different applications.
- *ExternalSingle*
Each object occupies a separate Natural session for its exclusive use.

Note:

Even though objects registered as ExternalMultiple can coexist with other objects in a Natural session, the multiple objects requested by one client are collected within one session. This means that objects from different clients can not interfere with each other.

Setting Activation Policies

The activation policy of a class can be set in three different ways, in the following order of precedence:

- Explicitly as part of the REGISTER command
- In the DEFINE CLASS statement
- With the profile parameter ACTPOLICY=*activation-policy* (for OS/390, DCOM=(ACTPOL=*activation-policy*))

When to use which Activation Policy

Non-trivial DCOM applications will mostly deal with *persistent* objects, i.e. objects stored in databases. For such applications, some considerations concerning database access, transaction handling and user isolation must be made. Consider the following scenario: clients A and B both create an object of a class that is provided by a certain NaturalX server process. Assume that the NaturalX server uses a database to load and store its objects. If both clients were served by the same server process, they would appear to the database as one single user. This would have the consequence that a transaction started by a method call from Client A can be committed or backed out by a method call from Client B. Such interferences are obviously to be avoided.

There are two approaches to avoiding this interference: either the clients do not use persistent objects, or each of them is served by its own NaturalX server process. Both approaches have their advantages in different situations; for a class or application that does not access databases or other shared resources, it is useful to serve several clients with a single server process. For classes that access databases or other shared resources, it is necessary to isolate different clients in different server processes. Hence both approaches should be possible. Activation policies give an administrator the means to control the activation behavior for each class at registration time.

Example

This example illustrates how the various activation policies can be used. Let us consider parts of an imaginary travel agency application. The application contains the business classes *Trip*, *Skipper* and *RoutePlanner*. The *Trip* class represents a sailing trip to be planned, the *Skipper* class represents the skippers available to lead the trips. *RoutePlanner* is a class that determines an optimal route for a trip. Assume that the *Trip* and *Skipper* classes use a database to read and store their objects. The *RoutePlanner* class just performs some calculations on a given *Trip* object and does not use a database.

Since some of the business classes use transactional access to a database, and a transaction might span several method calls, each active client needs to be served with its own NaturalX server process. This can be done by defining an additional class *SagTours*, which represents an application session. This class can be used, for example, to keep general information about the session status, but the main task will be to create business objects on behalf of a client.

Class SagTours

```
* Represents a SagTours application session.
*
define class SagTours
  local using tour-ids
  id clsid-sagtours
*
  interface Create /* Used to create application objects. */
    id iid-sagtours-create
*
  method newTrip /* Creates a new Trip object. */
    is trip-n
    parameter
    1 trip handle of object by value result
  end-method
```

```

    method newSkipper    /* Creates a new Skipper object. */
      is skip-n
      parameter
      1 skipper handle of object by value result
    end-method
  *
end-interface
*
end-class
end

```

This class will be registered as *ExternalSingle*. This means that each creation of a *SagTours* object starts a NaturalX server process for the client that requested the object. A client will create a *SagTours* object only once and will use its methods later to create the business objects it needs. In order to create a *Trip* object, the client will call the method *newTrip*, which is implemented as follows:

Method newTrip

```

  * This method creates a new Trip object.
  *
  define data parameter
    1 trip handle of object by value result
  end-define
  *
  create object trip of class "Trip"
  *
end

```

The *Trip* class itself will be registered as *InternalMultiple*. This ensures that the *Trip* objects created by the method *newTrip* are created in the NaturalX server process just started for this client.

Now let us look at the class *RoutePlanner*.

Class RoutePlanner

```

* Plans optimal routes for sailing trips.
*
define class RoutePlanner
  local using tour-ids
  id clsid-planner
*
  interface routing
    id iid-planner-routing
  *
  method plan    /* Plans a sailing trip. */
    is plan-n
    parameter
    1 trip handle of object by value
  end-method
*
end-interface
*
end-class
end

```

Method plan

```
* This method plans a sailing trip.
*
define data parameter
1 trip handle of object by value
end-define
*
* Perform some operations on the given Trip object.
*
end
```

This class can be registered as *ExternalMultiple*. In this case, all *RoutePlanner* objects created by different clients would be created in the same NaturalX server process. This does not do any harm if the methods of this class do not access databases, or if each database transaction is fully contained in a method (i.e. if each method subprogram ends with either a BACKOUT TRANSACTION statement or an END TRANSACTION statement).

Now let us look at a sample client program:

Sample Client Program

```
define data local
  sagTours handle of object
  trip handle of object
  planner handle of object
end-define
*
* Start the application session.
create object sagTours of "SagTours"
*
* Create a Trip object.
send "newTrip" to sagTours return trip
* Create a RoutePlanner object.
create object planner of "RoutePlanner"
* Plan the trip.
send "plan" to planner with trip
*
end
```

The client first creates a *SagTours* object. This starts a new NaturalX server process exclusively for this client. The client then uses the *SagTours* object to create a *Trip* object in the context of this application session. Note that the client creates the *RoutePlanner* object directly. This is possible because the class is registered as *ExternalMultiple*, but it is not necessary: the *SagTours* class could also provide a method for the creation of *RoutePlanner* objects. Afterwards it lets the business objects do their jobs. The objects are automatically released at program end. The deletion of the *SagTours* object causes the NaturalX server to shut down.

Note:

This example shows only the NaturalX techniques needed to illustrate the usage of activation policies. A real-world application would require a lot more. The classes would use object data areas and they would surely have globally unique IDs assigned. Also parameter data areas would be used instead of inline parameter declarations.

Registration

If a class is to be made accessible to DCOM clients, it is necessary to add some information about the class to the system registry. DCOM clients will mostly address a class with a meaningful name, the so-called programmatic identifier (ProgID) as in the following example:

```
CREATE OBJECT #01 OF CLASS "Employee"
```

For a Natural class, the class name defined in the DEFINE CLASS statement is written into the registry as a ProgID.

System registry entries map this ProgID to the globally unique ID (GUID) of the class, allowing DCOM to uniquely locate all information about the class. Further information that is stored in the registry includes the path and name of the responsible DCOM server, the path and name of the type library, and interface information.

This section covers the following topics:

- Registration with Natural
- Automatic Registration
- Manual Registration
- Registration Files and Type Library
- Client Registration
- Registration Hints

Registration with Natural

Natural classes can be registered (or unregistered) manually with the system command REGISTER(orUNREGISTER), automatically after the class is stowed (or deleted), or by running the .reg files which are generated every time a class is registered.

In order to register classes, you must have the rights to modify the system registry and your system environment must be able to use COM.

It is usually not advisable to change the Natural entries in the system registry directly in the registry editor because this can lead to inconsistent registry entries.

A class is always registered for the server ID under which Natural was started.

Note:

On the mainframe, the REGISTER command is only available under TSO and in batch.

Automatic Registration

If the profile parameter AUTOREGISTER(for OS/390 DCOM=(AUTOREG=*value*)) is set to ON, a Natural class is automatically registered when it is stowed (cataloged), and unregistered automatically when it is deleted. This means that the user can test the class directly after stowing it.

Automatic registration uses the activation policy setting defined in the WITH ACTIVATION POLICY clause of the DEFINE CLASS statement of the class. If this clause is not specified, the setting from the profile parameter ACTPOLICY=*activation-policy* (for OS/390, DCOM=(ACTPOL=*activation-policy*)) is used.

If automatic registration is set and a class is stowed (cataloged), the class is unregistered before it is stowed and registered after the stow has finished so that all old registry entries are removed.

Manual Registration

The REGISTER Command

The system command REGISTER is used to register Natural classes. They are registered for the server ID under which Natural was started.

$\text{REGISTER } \left\{ \begin{array}{c} \text{class-module-name} \\ * \end{array} \right\} \left[\begin{array}{c} \left\{ \begin{array}{c} \text{library-name} \\ * \end{array} \right\} \left[\begin{array}{c} ES \\ IM \\ EM \end{array} \right] \end{array} \right]$
--

Note:

On the mainframe, the REGISTER command is only available under TSO and in batch.

class-module-name

This defines which class or classes are to be registered by specifying the appropriate Natural object module name.

library-name

This defines which library or libraries are to be searched for the class or classes.

ES IM EM

This defines the activation policy which is registered for the class or classes.

You can set one of the following parameters:

Parameter	Description
ES	Sets activation policy <i>ExternalSingle</i>
IS	Sets activation policy <i>InternalSingle</i>
EM	Sets activation policy <i>ExternalMultiple</i>

The following table shows which classes will be registered for all possible class/library combinations:

Class Module Name Specification	Library Name Specification		
<i>library-name</i>	*	-	
<i>class-module-name</i>	class with class module name <i>class-module-name</i> of library <i>library-name</i>	all classes with the class module name <i>class-module-name</i> which are found in the current step libraries	class with class module name <i>class-module-name</i>
*	all classes which are found in the library <i>library-name</i> are registered	all classes which are found in the current step libraries are registered	all classes of the current logon library are registered

If this parameter is not specified in the REGISTER command or the DEFINE CLASS statement, the default activation policy defined in NATPARM is used.

The UNREGISTER Command

The system command UNREGISTER is used to unregister Natural classes.

```
UNREGISTER { class-module-name } [ { library-name } [ server-ID ] ]
```

class-module-name

This defines which class or classes are to be unregistered by specifying the appropriate Natural object module name.

library-name

This defines the library or libraries which are to be searched for the class or classes.

server-ID

This defines the server ID of the class or classes.

The following table shows which classes will be unregistered for all possible class/library/server ID combinations:

Class Name Specification	Library Name /Server ID Combination				
	- -	<i>library-name</i> -	<i>library-name</i> <i>server-ID</i>	* -	* <i>server-ID</i>
<i>class-module-name</i>	class with <i>class-module-name</i> in the current logon library if it is registered for the current server ID	class with <i>class-module-name</i> of library <i>library-name</i> if it is registered for the current server ID	class with <i>class-module-name</i> of library <i>library-name</i> if it is registered for the server <i>server-ID</i>	all classes with <i>class-module-name</i> found in the current step libraries if they are registered for the current server ID	all classes with the name <i>class-module-name</i> found in the current step libraries which are registered for the server <i>server-ID</i>
*	all classes of the current logon library which are registered for the current server ID	all classes found in the library <i>library-name</i> which are registered for the current server ID	all classes found in the library <i>library-name</i> which are registered for the server <i>server-ID</i>	all classes found in the current step libraries which are registered for the current server ID	all classes found in the current step libraries which are registered for the server <i>server-ID</i>

A REGISTER or UNREGISTER system command will return an error message if *class-module-name* or *class-module-name* and *library-name* are specified but either the class or library is not found. If only an asterisk (*) is given in the REGISTER or UNREGISTER system command, no error message is returned if no class has been registered or unregistered.

If a class without class GUIDs or interface GUIDs is specified in the REGISTER system command, an error message will be returned. Such a class can only be used in the local Natural session.

Registration Files and Type Library

Registration files (.reg files) enter information in the system registry when they are executed.

Natural will automatically create registration files for the server and the client side when a class is registered.

The server .reg file contains the same information that was entered in the system registry and the client .reg file contains all information which is generated for the client side. When a class is unregistered, the .reg files will be deleted. If a .reg file is not to be deleted with the unregistration, the file has to be renamed before unregistering the class because Natural deletes only files with the default .reg file names.

The .reg files will be named <classmodule_name>_S.reg (for the server) and <classmodule_name>_C.reg (for the client) and, to activate a different version, <classmodule_name>_V.reg.

A type library is created automatically when a class is registered, and it is deleted when a class is unregistered. A reference to the type library is also entered in the registry.

The default type library name is <classmodule_name>.tlb. A new name will be generated if a type library with this name exists already.

The registration files and the type library are stored in the Natural *etc*-directory as follows:

```
$NATDIR/$NATVERS/etc/<serverid>/<classname>/v<version-number>
```

Example

The files for version one of a class MY.TEST.CLASS registered for the server ID SERVER01 are located as follows:

```
$NATDIR/$NATVERS/etc/SERVER01/MY.TEST.CLASS/v1
```

Client Registration

Natural does not enter the registration information for the clients automatically in the system registry, but creates a registration file for the client. The client registration file contains an entry (RemoteServerName) that tells DCOM on which machine the DCOM server class can be found. This entry is not filled from Natural. It can be entered in either of two ways:

1. The RemoteServerName can be entered in the registration files. In this case the line

```
"RemoteServerName"=
```

has to be changed to

```
"RemoteServerName"="<server_machine_name>"
```

After this, the registration file has to be executed on the client machine.

2. The registration file is executed first, and then the RemoteServerName is changed using the DCOMCNFG tool or the Registry Editor (see the section Configuration Examples).

Registration Hints

The following points should be taken into account when registering and unregistering classes:

- The class GUID should never be changed for an existing class: Natural displays an error message if a class that is already found in the registry is registered again with another GUID. The old class must first be unregistered in this case.
- The same class should never be registered for more than one server ID: there is a one-to-one relationship between the server ID and the AppID, and a class has only one AppID defined, which means that a registration for a second server ID overwrites the AppID. Furthermore, if the class is unregistered for one server ID, all entries of the class are removed without checking whether it is registered for a second server.
- Except for client registration, you should always use the Natural system commands REGISTER and UNREGISTER to change registry entries for a class because they remove redundant registry entries. For example, if a client class has been registered for server1 and a server registration file with a registration of the same class for server2 is run, the AppID key of the class is changed and all references to the old AppID key are lost. So this old AppID key can never be deleted. When a class is registered with the system command REGISTER, a check is made to see whether the AppID has been changed, and the old AppID is removed if no other class needs it.
- If Natural is not available on the client machine and registry entries for a Natural class are to be removed from the system registry, you should do this with the registry editor. If Natural is available on the client machine, it is easier to register the class first with the Natural system command REGISTER and unregister it afterwards with the system command UNREGISTER.
- The registration information for a class is taken from the catalogued class object, so that it is not possible to register or unregister a class that is only available in source format.
- If you want to register classes during a Natural session, the session must be started with the parameters PARM and COMSERVERID=*server-ID* (for OS/390, DCOM=(SERVID=*server-ID*)) only as shown below. This is because only these two parameters are stored in the registry key "LocalServer32". If a class is tested with other parameter settings, there is no guarantee that it will run later when it is started from a DCOM client.

Windows 98/NT/2000:

```
NATURAL.EXE PARM=COMPARM COMSERVERID=SERVER1
```

UNIX:

```
NATURAL PARM=COMPARM COMSERVERID=SERVER1
```

OS/390:

```
NATURAL DCOM=(SERVID=SERVER1)
```

- Session Parameters for NaturalX Servers (OS/390 Only): If DCOM launches the server, the session parameters defined in the system registry are used. Although the REGISTER command only sets the parameter DCOM=(SERVID=*server-id*), it is possible to maintain the server session parameters in the system registry using the Natural system command DCOMPARM. If you start the server manually from the shell, you can specify dynamic session parameters with the shell command. For example:

```
naturalx "profile=myprofile fnat=(10,930) dcom=(serverid=employees)"
```

If you start the server manually in batch/TSO, you can specify dynamic session parameters with the "naturalx" command or via the dataset name CMPRMIN.

- Usually only users with administrator rights can change the system registry. So if you receive an error when trying to register a class, check to see whether you have the rights required to change the registry.
- When a Natural class is registered, some additional information is entered in the registry that is only needed by Natural (not by DCOM). The information which is stored in the additional registry keys is the server ID (see section NaturalX Servers), the activation policy (see section Activation Policies) and the location (Natural class module name and library of class) of the class. This information is necessary, for example, if all classes of a specified server ID are to be unregistered or to make the served classes available when Natural is started.

- There is a one-to-one relationship between the server ID and the AppID (under HKEY_CLASSES_ROOT/AppID) of a class. When a class is registered for a new server ID, a new GUID - the AppID, is generated and assigned to this server ID. The AppID is used by DCOM to group the DCOM classes. Security settings and (for client registrations) the remote machine name are defined for an AppID, i.e. all classes which belong to one AppID have the same security settings (see the sections Configuration Overview and Security).

DCOMPARM System Command - OS/390 Only

The system command DCOMPARM is used to display and modify the Natural parameters for a specified server ID in the system registry.

DCOMPARM [*server-ID*]

The parameters for the specified server ID are read from the system registry and are displayed in the 'DCOM Parameters' screen, where they can be modified. Make sure that no parameter is split at the end of an input line.

When you save your changes, NaturalX searches the parameter list for the parameter DCOM=(SERVERID=*serverID*). If the parameter is found, it is moved to the end of the list. If it is not found, it is created at the end of the parameter list. No other data validation takes place.

You can reread the parameters from the system registry for the specified server ID by issuing the READ command.

server-ID

Specify the server ID which was used for the Natural session in which the REGISTER command for the Natural class was executed.

Note:

You can find the server ID in the system registry.

Type Information

This section covers the following topics:

- Overview
- NaturalX and Type Information
- Using Type Information

Overview

Type information is a means to completely describe a class along with all of its interfaces, down to the names and types of the methods. It contains the necessary information about classes and their interfaces, for example, which interfaces exist on which classes, which member functions exist in those interfaces, and which argument those functions require.

This information is used by clients to find out details about a class and its methods, for example, by type-information browsers to present available objects, interfaces, methods and properties to an end user.

Another important area for using type information is the widely-used OLE automation technique which is also used by NaturalX.

There are several ways to store type information. A common way is generating the type information in type library (.TLB) files.

NaturalX and Type Information

Creating Type Information

For each Natural class, a type library file is created when the class is registered.

The type library is generated in the \$NATDIR/\$NATVERS/etc/<serverid>/<classname>/<version> directory and connected to the class via an entry in the registry.

The name of the class module is used, and the .tlb extension is appended unless the type library file name conflicts with an existing name. Then a number is attached to the class module name.

Using Type Information

Each interface defined in a Natural class is seen by clients as a dynamic interface (also called a dispatch interface). Each method of an interface is seen by clients under the name defined in the DEFINE METHOD statement.

The first interface in a Natural class is marked as the default dispatch interface.

The support of type information also makes it possible to define multiple interfaces with identical method/property names. The Natural client simply addresses the corresponding method by using the interface name (as defined in the Natural class) as the prefix of the method name, as shown in the following example:

```
CREATE OBJECT #03 OF CLASS "DepartmentList"  
  SEND "Iterate.PositionTo" TO #03 WITH "C" RETURN #DEPT
```

Natural clients use type information to find out to which interface a method or property belongs.

Note:

Natural clients do not use type information at catalog time to perform syntax checks.

Data Type Conversions

Natural Types to OLE Types

In order to receive data from clients or to pass data to classes written in different programming languages, the Natural data types are converted to so-called OLE automation-compatible types. This table shows how clients see method parameters or properties of a Natural class. For example, if a Natural class has a method parameter or a property with the format A, this is seen by clients as VT_BSTR.

Natural Data Type	Automation-Compatible Type
A	VT_BSTR
B1	VT_UI1
B2	VT_UI2
B4	VT_UI4
B n ($n \neq 1, 2, 4$)	SAFEARRAY of VT_UI1
C	not supported
D	VT_DATE
F4	VT_R4
F8	VT_R8
I1	VT_I2
I2	VT_I2
I4	VT_I4
HANDLE OF GUI	not supported
HANDLE OF OBJECT	VT_DISPATCH
L	VT_BOOL
N15.4	VT_CY
N $n.m$ ($n.m \neq 15.4$)	VT_R8
P15.4	VT_CY
P $n.m$ ($n.m \neq 15.4$)	VT_R8
T	VT_DATE

An array of a given Natural type is mapped to a SAFEARRAY of the corresponding VT type.

There are, however, some special cases:

- A variable of type B n , where n is not 1, 2, 4 or an array of such types, is always mapped to a one-dimensional SAFEARRAY of VT_UI1. This is the way recommended by Microsoft to transport binary data through a dispatch interface.
- Control variables are not mapped. They have no meaning outside of Natural. Variables of type HANDLE OF GUI are also not mapped. There is no corresponding automation-compatible type. Therefore properties of the type Control variable or HANDLE OF GUI cannot be accessed by clients through COM/DCOM. Method parameters of these types should be marked as optional in the parameter data area, so that clients can omit the parameters when calling the method through COM/DCOM.

OLE Types to Natural Types

This table shows how parameters or properties of an external class can be addressed by Natural. For example, if an external class has a method parameter or property with format VT_R4, this parameter or property can be addressed in Natural as F4 or with a format that is MOVE-compatible to F4.

Automation -Compatible Type	Natural Data Type
VT_BOOL	L
VT_BSTR	A
VT_CY	P15.4
VT_DATE	T
VT_DISPATCH	HANDLE OF OBJECT
VT_UNKNOWN	HANDLE OF OBJECT
VT_I1	I1
VT_I2	I2
VT_I4	I4
VT_INT	I4
VT_R4	F4
VT_R8	F8
VT_U1	B1
VT_U2	B2
VT_U4	B4
VT_UINT	B4

A SAFEARRAY of up to three dimensions is converted into a Natural array with the same dimension count and the corresponding format. SAFEARRAYs with more than three dimensions cannot be used from within Natural.

There are, however, some special cases:

- A VT_BSTR maps either to a Natural Alpha variable or to a one-dimensional array of Natural Alpha variables. The additional dimension is used to store strings longer than 253 characters.
- A SAFEARRAY of VT_BSTRs maps either to an array of Natural Alpha variables with the same dimension count, or to an array with one more dimension. The additional dimension is used to store strings longer than 253 characters.
- A SAFEARRAY of VT_UI1 can be mapped to any array of Natural binaries that has a matching total size. This is because a SAFEARRAY of VT_UI1 is the way to transport binary data through a dispatch interface.

Configuration Overview

Once all classes of an application have been registered on the client and server machines, certain aspects of the application's behavior can be controlled and configured with system registry settings. This section summarizes the relevant registry entries and their meaning for NaturalX applications. For detailed background information about the registry keys and their administration, please refer to the specific DCOM registry documentation of the appropriate platform.

The registry keys relevant in this context are maintained with commonly-used tools like DCOMCNFG or the Registry Editor (REGEDIT). These tools present the registry keys in a different way. Therefore only the names of the registry keys are mentioned here. The section Configuration Examples describes how to set registry keys.

Note:

HKLM is the common short form of the registry key HKEY_LOCAL_MACHINE, where HKCR stands for HKEY_CLASSES_ROOT.

This section covers the following topics:

- Server Configuration - General Settings
- Server Configuration - Application-Specific Settings
- Client Configuration - General Settings
- Client Configuration - Application-Specific Settings

Server Configuration - General Settings

This section discusses general server configuration settings.

- The registry entry *HKLM\Software\Microsoft\OLE\EnableDCOM* must be set to 'Y' to enable access to the server machine via DCOM.
- If guests (users who do not have their own account on the server machine) are to be able to access applications on the server machine, the predefined account *Guest* must be enabled in the User Manager (Windows NT and Windows 2000 only).
- The registry entries *HKLM\Software\Microsoft\OLE\DefaultLaunchPermissions* and *HKLM\Software\Microsoft\OLE\DefaultAccessPermissions* define which users or groups are allowed or not allowed to launch DCOM applications and to access their classes. The authorizations defined here apply for all applications for which no application-specific settings are defined.
- The registry entry *HKLM\Software\Microsoft\OLE\LegacyAuthenticationLevel* controls the level of authentication that is performed for clients that access DCOM applications on this machine. If a NaturalX server is to be able to pass the client's user ID to Natural Security, the setting should be at least *Connect*. Choose *None* if no authentication is to take place. In this case, the NaturalX server does not retrieve the client's user ID. Instead it performs each request under the user ID under which it was launched. If this entry is defined differently on the client side and on the server side, the stricter setting applies.

- The registry entry *HKLM\Software\Microsoft\OLE\LegacyImpersonationLevel* controls how much information a server may retrieve about the client, or if it may even use this information to act in the role of the client against other servers. If a NaturalX server is to be able to pass the client's user ID to Natural Security, the setting should be at least *Identify*. The settings *Impersonate* or *Delegate* have the same effect for a NaturalX server. Choose *Anonymous*, if the server is not to be able to retrieve the client's user ID. In this case, the server performs each request under the user ID under which it was launched. If this entry is defined differently on the client side and on the server side, the stricter setting applies.

Server Configuration - Application-Specific Settings

The application-specific settings can be set up differently for each NaturalX application. But the question is where to apply these settings. It is important to remember that all classes registered under one NaturalX server ID form one application in the DCOM sense, and are thus assigned to one AppID key in the registry. This is why the application-specific settings are applied under the AppID key.

- The registry entries *HKCR\AppID\<APPID>\LaunchPermission* and *HKCR\AppID\<APPID>\AccessPermission* define which users or groups are allowed or not allowed to launch the DCOM application with the specified AppID and to access its classes.
- The registry entry *HKCR\AppID\<APPID>\RunAs* defines the user account this NaturalX server will run when it is launched by DCOM. There are three options:
 - *Interactive user:*
The NaturalX server is started under the account of the user that is interactively logged in on the server machine. This is usually not desirable but can be useful for test reasons.
 - *Launching user:*
The NaturalX server is started under the account of the client that creates the first object on this server (remember that the first request for an object forces DCOM to launch the server). This setting should be used if each client is to be served by its own server process. Obviously, the client must have permission to launch the server.
 - *This user:*
The server is started under the account of a given user. This setting should be used if all clients are to be served by the same server process. The user entered here must have permission to launch the server.

Client Configuration - General Settings

This section discusses general client configuration settings.

- The registry key *HKLM\Software\Microsoft\OLE\LegacyAuthenticationLevel* controls the degree of authentication that is performed for clients running on this machine when they access DCOM applications. For a client that accesses a NaturalX server, a similar consideration to that in the section *Server Configuration - General Settings* applies: only if it specifies at least *Connect*, will the NaturalX server be able to use its user ID against Natural Security. If this entry is defined differently on the client side and on the server side, the stricter setting applies.
- The registry key *HKLM\Software\Microsoft\OLE\LegacyImpersonationLevel* controls how much information a server may retrieve about the client, or if it may even use this information to act in the role of the client against other servers. For a client that accesses a NaturalX server, a similar consideration to that in the section *Server Configuration - General Settings* applies: only if it specifies at least *Identify*, will the NaturalX server be able to retrieve its user ID and use it against Natural Security. If this entry is defined differently on the client side and on the server side, the stricter setting applies.

Client Configuration - Application-Specific Settings

The application-specific settings can be set up differently for each NaturalX application. But the question is where to apply these settings. Remember that all classes registered under one NaturalX server ID form one application in the DCOM sense, and are thus assigned to one AppID key in the registry. This is why the application-specific settings are applied under the AppID key.

- The registry key *HKCR\AppID\<APPID>\RemoteServerName* defines on which remote machine DCOM should start the server when a class hosted by this server is requested. If the server is to be started locally, 'Run on this computer' and no *RemoteServerName* must be specified.

Sample Application

On Windows 98/NT/2000, a sample application is provided in the library SYSEXNXX. For information on how to run this application, see the A-README member in the library SYSEXNXX.

Security with NaturalX

This section covers the following topics:

- Overview
- Activation Security
- Call Security

Information on how to configure NaturalX is given in the section Configuration Examples.

Overview

In a distributed environment, security is an especially important topic. A server must be sure that no unauthorized clients use the services it provides. A client must be sure that it is connected to the server it expects, and that the server does not misuse its (the client's) authorizations.

In the context of DCOM, two levels of security can be distinguished:

- Activation security controls who is allowed to launch and access the server process that provides the class.
- Call security controls who is allowed to use the individual methods a class provides.

In many cases, activation security may be sufficient to define authorizations. This security level is supported by DCOM itself on the basis of Windows Security. The necessary authorizations are maintained in the system registry. This is described in the section Activation Security.

In other cases it may be necessary to control authorizations in more detail at the level of individual methods. This security level cannot be maintained with registry definitions. It is, therefore, provided by NaturalX with the help of Natural Security. This is described in the section Call Security.

Activation Security

This section covers the following topics:

- Applications
- Authorizations using the Registry

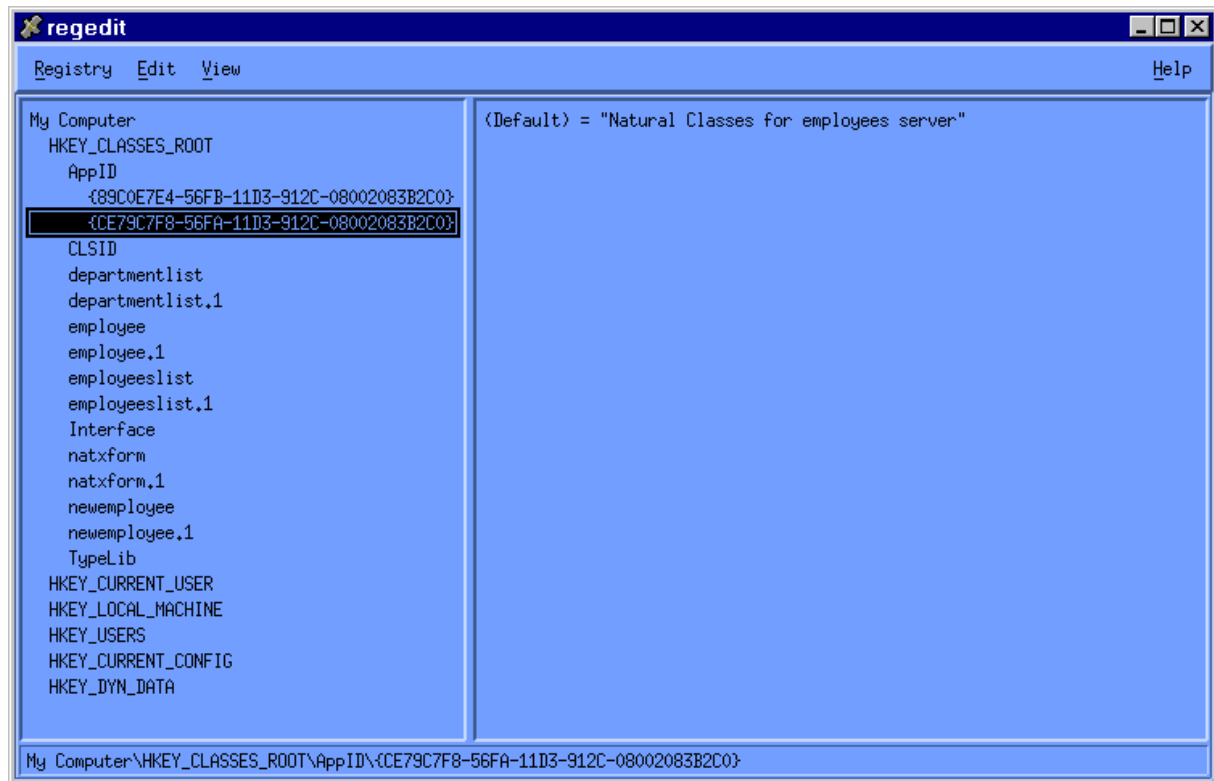
Applications

Activation security controls who is allowed to launch and access a server process. In principle, this could be done by defining authorizations for each individual class. For practical reasons, however, and to reduce administration efforts, authorizations are normally maintained at the application level. In the system registry, each application is defined by an AppID. The AppID is the key under which the authorizations for an application are maintained. To maintain these authorizations, each DCOM enabled platform provides the tool DCOMCNFG. This tool can be used for NaturalX applications as well as for other DCOM applications.

In order to understand the meaning of AppIDs in NaturalX, recall for a moment how NaturalX organizes classes to applications (see the section Organizing Server IDs). With the Natural parameter COMSERVERID=*server-ID* (for OS/390, DCOM=(SERVID=*server-ID*)), a name can be given to a certain NaturalX server. When Natural is started with a given value of COMSERVERID=*server-ID* (for OS/390, DCOM=(SERVID=*server-ID*)), all Natural classes that are registered during this Natural session are registered under this server ID. At the same time, they are all registered under the same AppID key in the system registry. This means that each different value of *server-ID* corresponds to a different AppID key in the system registry.

As an example, assume Natural is running with the server ID "Employees". All classes registered during this Natural session will then form the "Employees" application. The REGISTER command registers them all under one AppID key - the one that corresponds to the "Employees" application.

In the Registry Editor of EntireX DCOM for example, the application ID will appear as follows:



Authorizations using the Registry

When configuring Activation Security, the following registry keys are of interest: *LaunchPermissions*, *AccessPermissions*, *DefaultLaunchPermissions* and *DefaultAccessPermissions*. The keys *DefaultLaunchPermissions* and *DefaultAccessPermissions* exist only once in the registry and define authorizations for all applications for which no individual authorizations have been defined. The keys *LaunchPermissions* and *AccessPermissions* exist for each application (i.e. for each AppID) and define the authorizations for an individual application.

Call Security

This section covers the following topics:

- Authorizations using Natural Security
- Security Hints and Suggestions

Authorizations using Natural Security

Call security is used to control who is allowed to use the individual methods that a class provides. Authorizations on this level cannot be maintained by registry definitions. Call security is therefore provided by NaturalX with the help of Natural Security.

In order to understand how call security is achieved with Natural Security, consider how a class in NaturalX is implemented: each class is a Natural module of type class, each method is a Natural module of type subprogram. For all Natural modules, the execution can be controlled by authorizations defined in Natural Security. Please refer to the *Natural Security Manual* for further information about how to do this.

The authorizations defined for class modules and method subprograms are evaluated whenever a class module is used to create objects and whenever a method subprogram is executed in response to a method call. The following rule applies: a user who is allowed to execute the class module is allowed to create objects of that class, and a user who is allowed to execute a method subprogram is allowed to use the corresponding method.

In order to perform the necessary authorization checks, a NaturalX server must know the client's user ID. It must also be sure that the user ID is authentic. Therefore the following requirements must be met to use call security:

- The client must have identified itself with a logon on its local machine or on a Windows domain server.
- *Authentication level* must be set to at least *Connect* (either on the client or on the server machine).
- *Impersonation level* must be set to at least *Identify* (either on the client or on the server machine).

If the above requirements are met, a NaturalX server that is going to process a request takes the client's user ID and places it into the Natural system variable *USER. The request is then performed under this user ID, including all necessary Natural Security authorization checks. After having processed the request, the Natural system variable *USER reverts to the value that it had at the startup of the NaturalX server.

If one of the requirements is not met, *USER remains unchanged during execution of the request. The request is then executed under the user ID under which the NaturalX server was started.

In addition to *USER, also the system variable *NET-USER is filled during execution of a request. It contains the user ID qualified with the domain name for clients belonging to a Windows domain and can be used for additional application-specific security checks.

Security Hints and Suggestions

The following points should be taken into consideration when using NaturalX with Natural Security:

- In a Natural Security environment, a NaturalX server must be started with the Natural parameter AUTO=ON. This is because the authentication already takes place on the client side. The setting should be entered in the Natural parameter module.
- In a Natural Security environment, it is a good idea to let a NaturalX server always start under a specific user ID. This user ID is then automatically used for all requests of unauthenticated users, and it is up to the Natural Security administrator to define minimal authorizations for this user ID.
- Remember that Natural and Natural Security cannot handle user IDs which are longer than 8 characters or which contain blanks.

NaturalX Configuration Examples

This section gives you an example of how to configure DCOM for NaturalX on each available platform, providing examples for both client and server configuration.

This section covers the following topics:

- DCOM Configuration on Windows NT/2000
 - DCOM Configuration on Windows 98
 - DCOM Configuration on Windows 98 in a Windows NT Domain
 - DCOM Configuration on UNIX
 - DCOM Configuration on OS/390
-

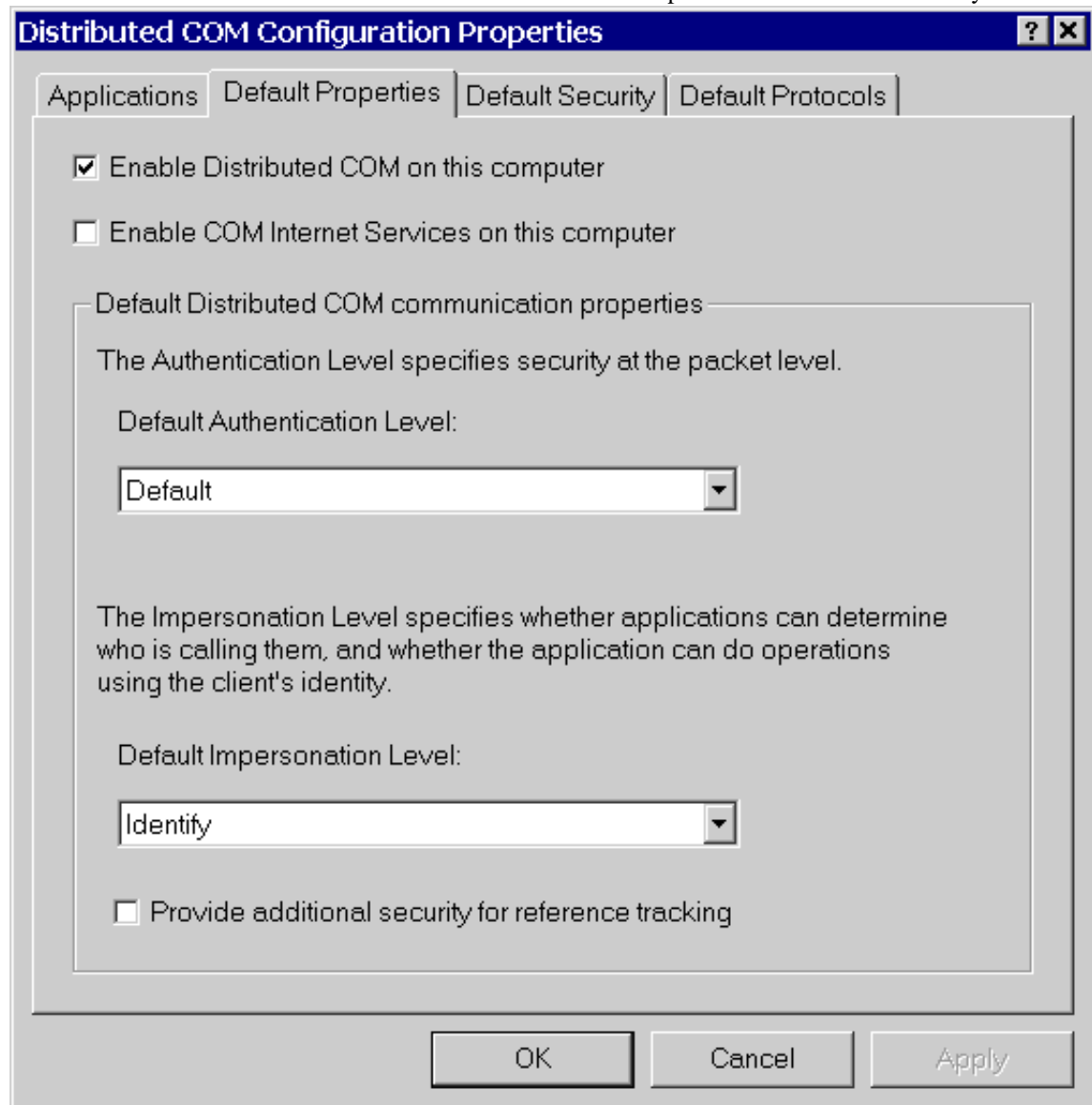
DCOM Configuration on Windows NT/2000

This section describes how to configure NaturalX applications on Windows NT and Windows 2000. All settings are applied with the tool DCOMCNFG.

- Configuring NaturalX Servers on Windows NT/2000
- Configuring NaturalX Clients on Windows NT/2000

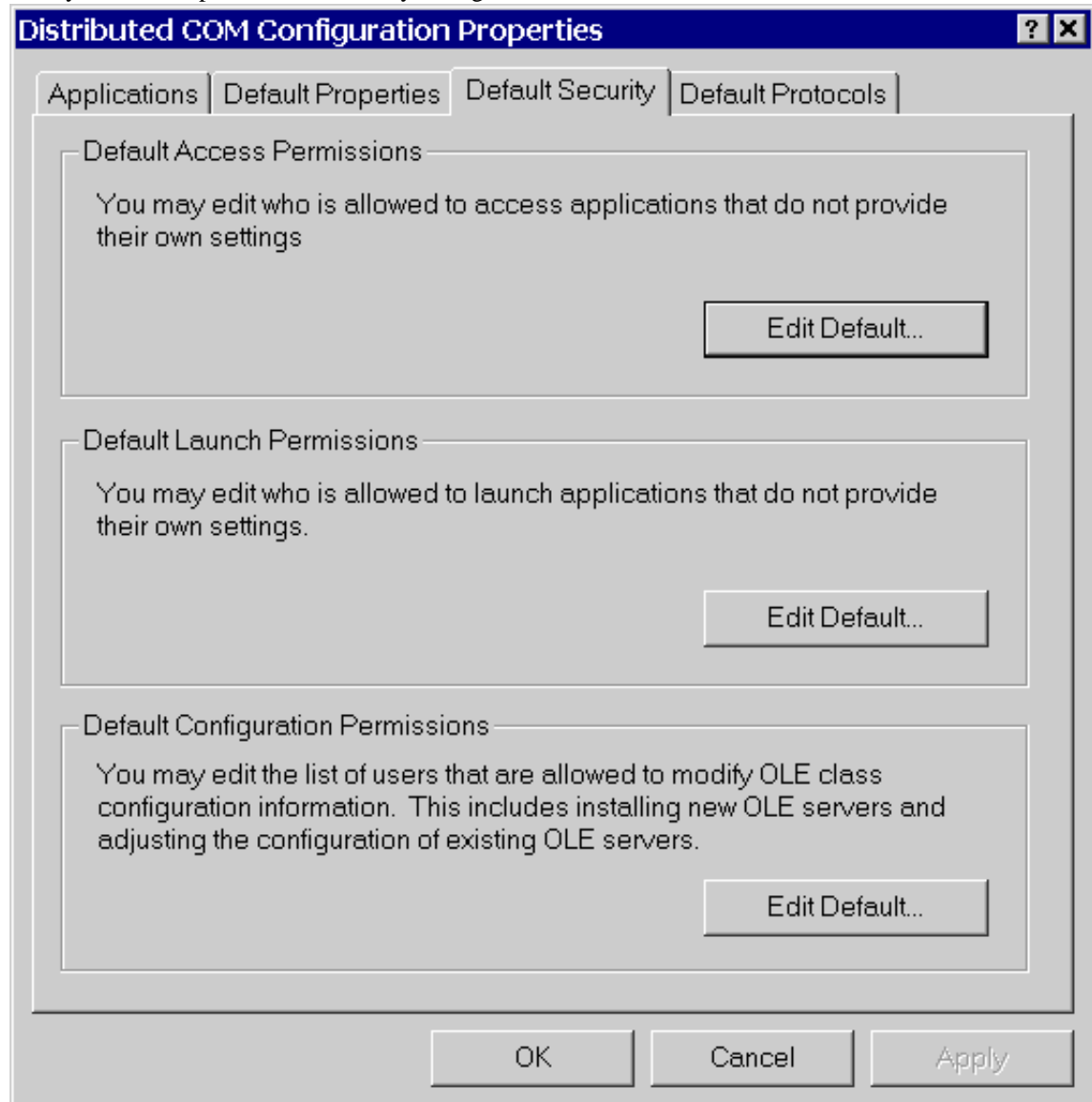
Configuring NaturalX Servers on Windows NT/2000

1. Invoke the Distributed COM Configuration Properties dialog box.
2. In the "Default Properties" tab, activate the checkbox "Enable Distributed COM on this computer".
3. Set "Default Authentication Level" to "Default" and "Default Impersonation Level" to "Identify".



This allows NaturalX servers to retrieve the client's user ID. Before executing a request, the server will then move the client's user ID into the Natural system variable *USER in order to let Natural Security checks run against this user ID.

Now you can set up the default security configuration.



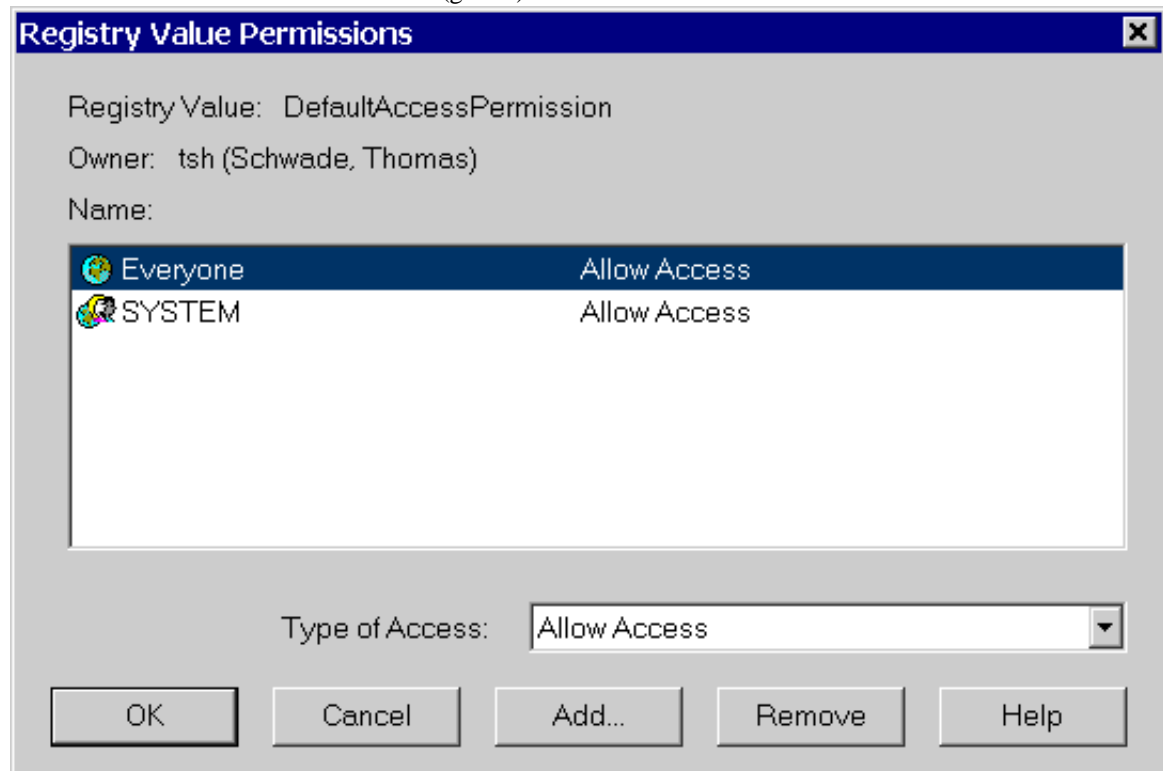
4. In the "Default Security" tab, choose Edit Default in the Default Access Permissions box. The Registry Value Permissions dialog box appears.

5. Use the Add function to define which users and groups can access NaturalX servers.

Note:

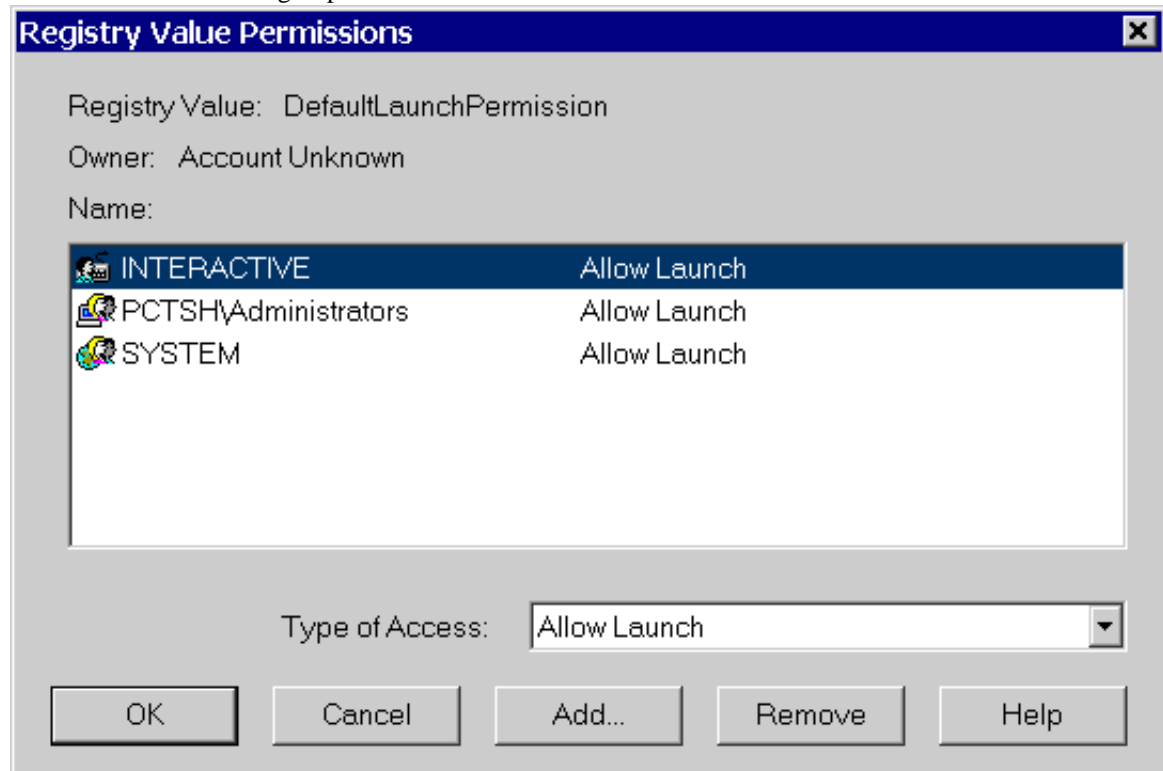
The registry value "DefaultAccessPermission" must contain at least the account SYSTEM.

In most cases you will define a group of all users to whom you want to grant access and enter this group here. In the example, the built-in group "Everyone" is entered. This grants access to every user that is defined on the server machine. If the built-in account "Guest" is enabled in the User Manager, this setting grants access to users not defined on the server machine (guests) as well.



6. Edit the default launch permissions.

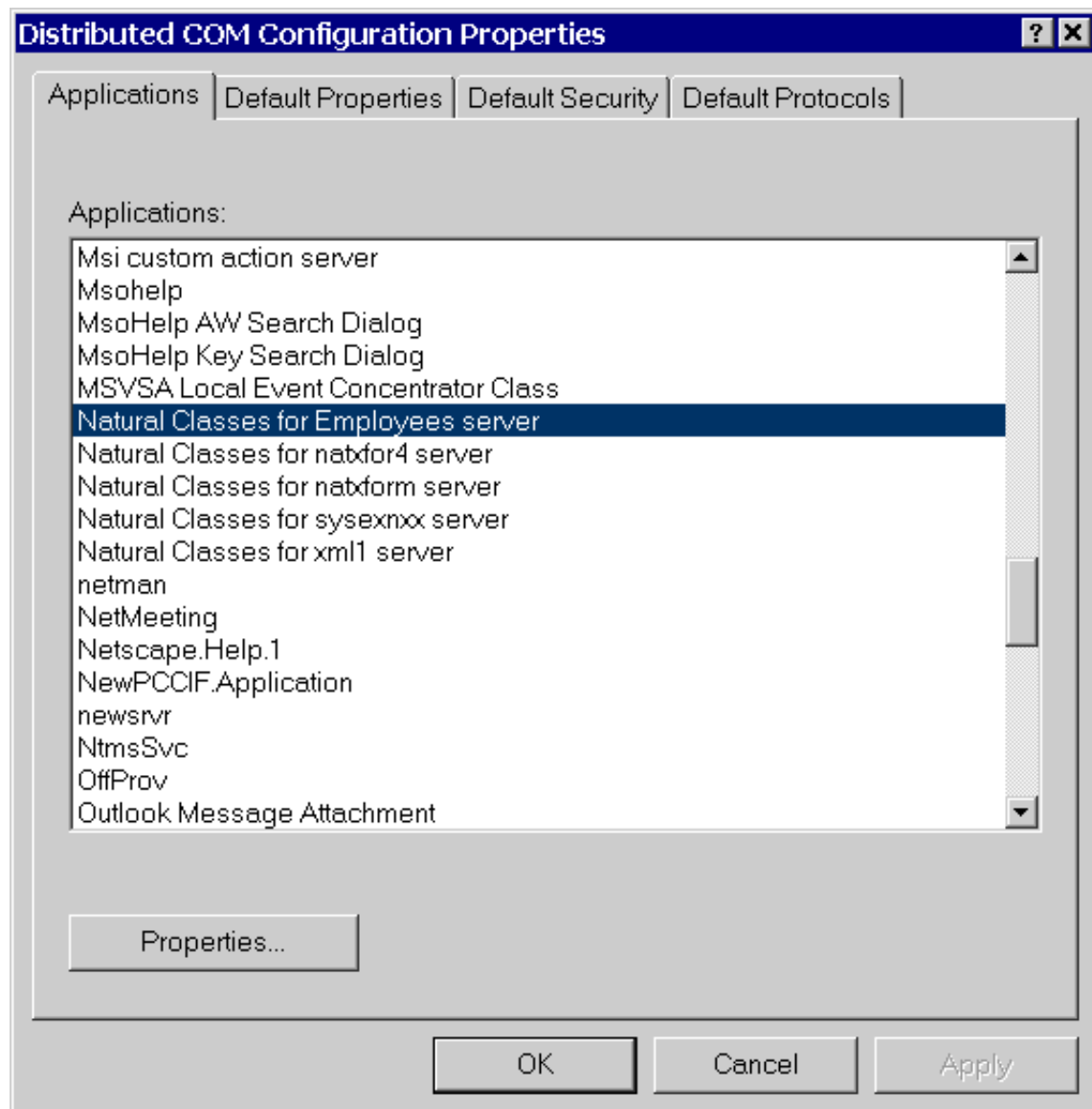
The registry value "DefaultLaunchPermission" must contain at least the accounts SYSTEM and INTERACTIVE and the group Administrators.



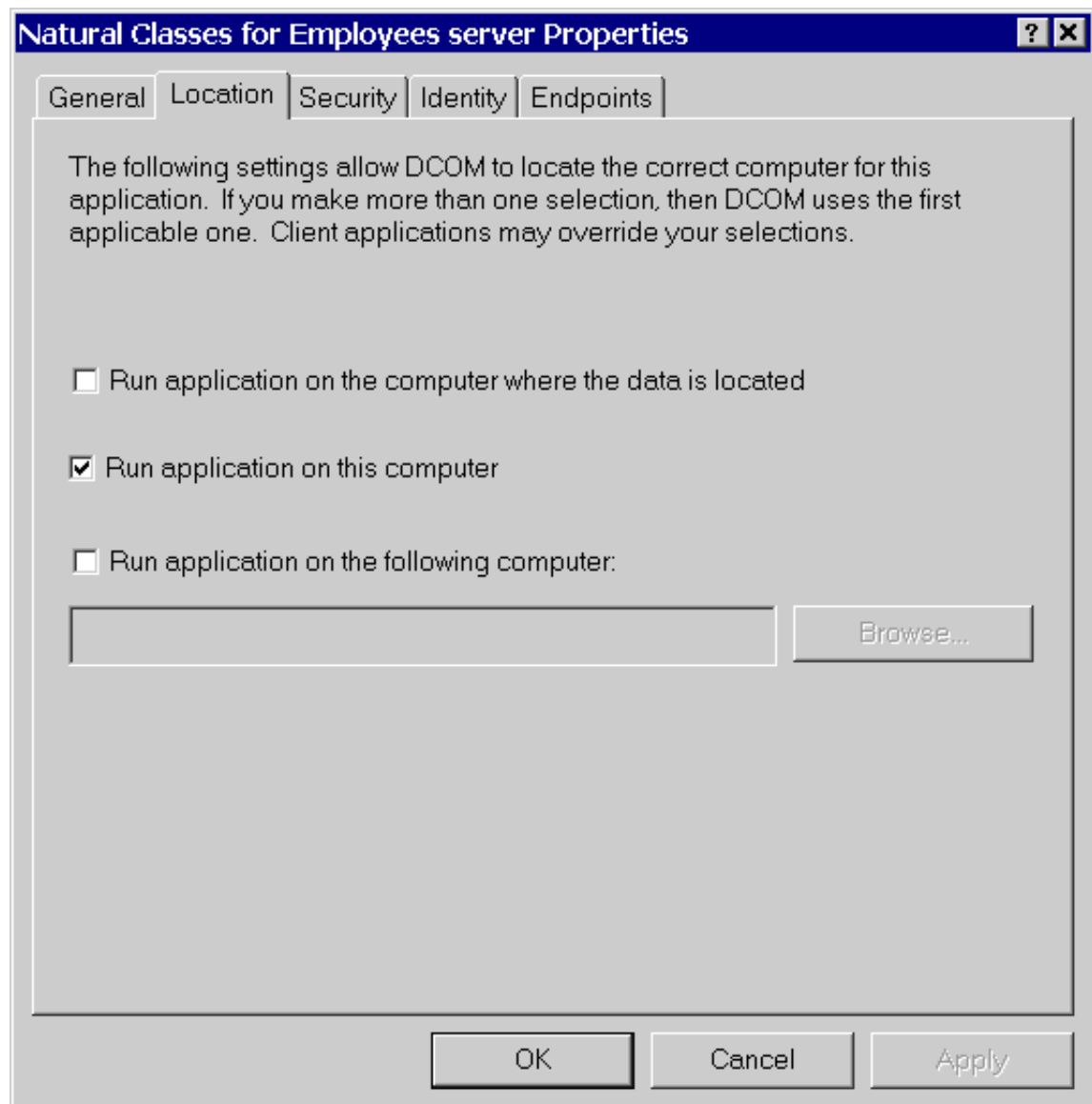
Now you can set up the configuration for a specific NaturalX server.

7. In the "Applications" tab, locate your NaturalX server in the Applications list box (in the example "Natural classes for Employees server").

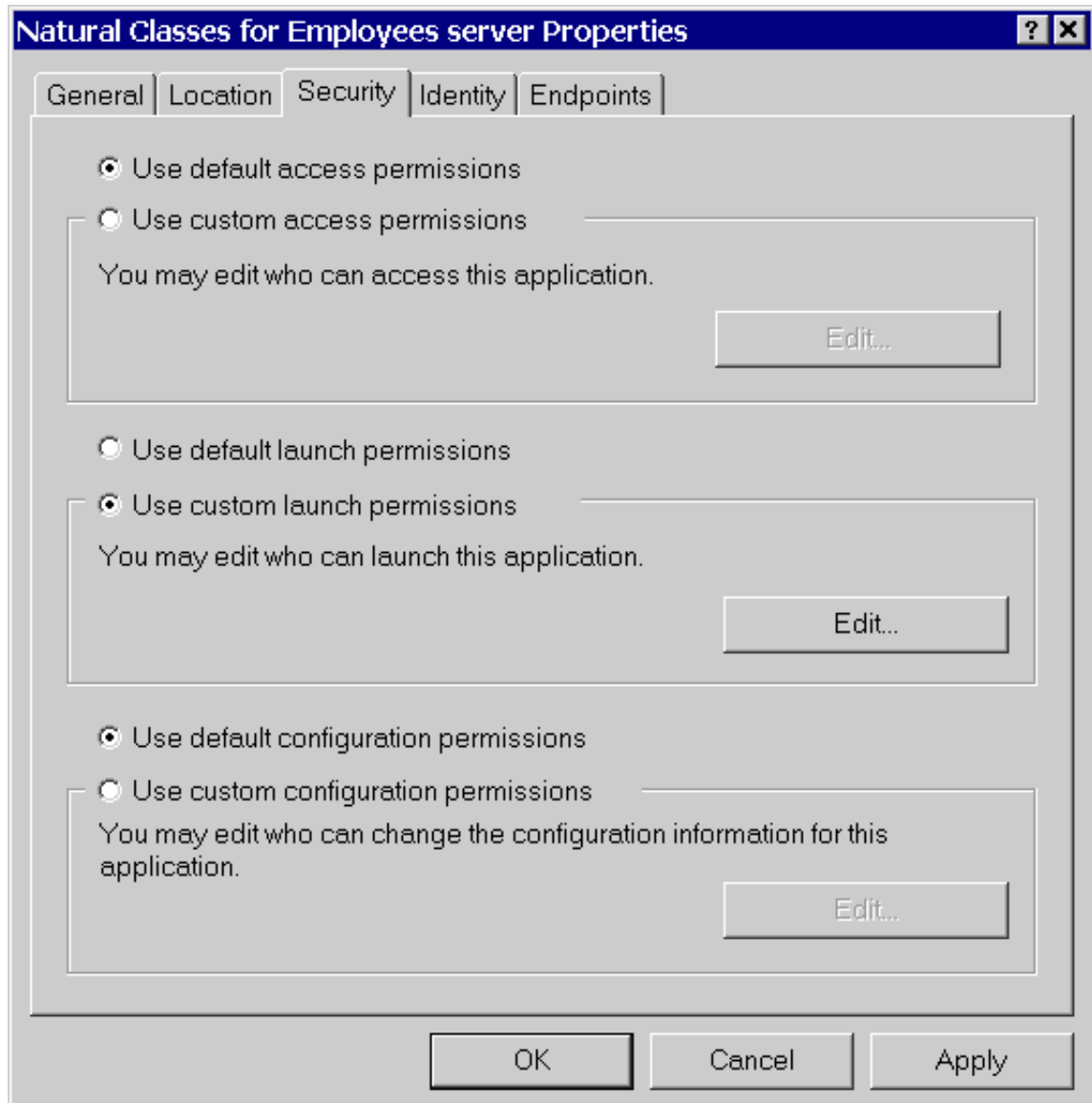
8. Select your server and choose "Properties".



9. In the "Location tab", activate the checkbox "Run application on this computer".



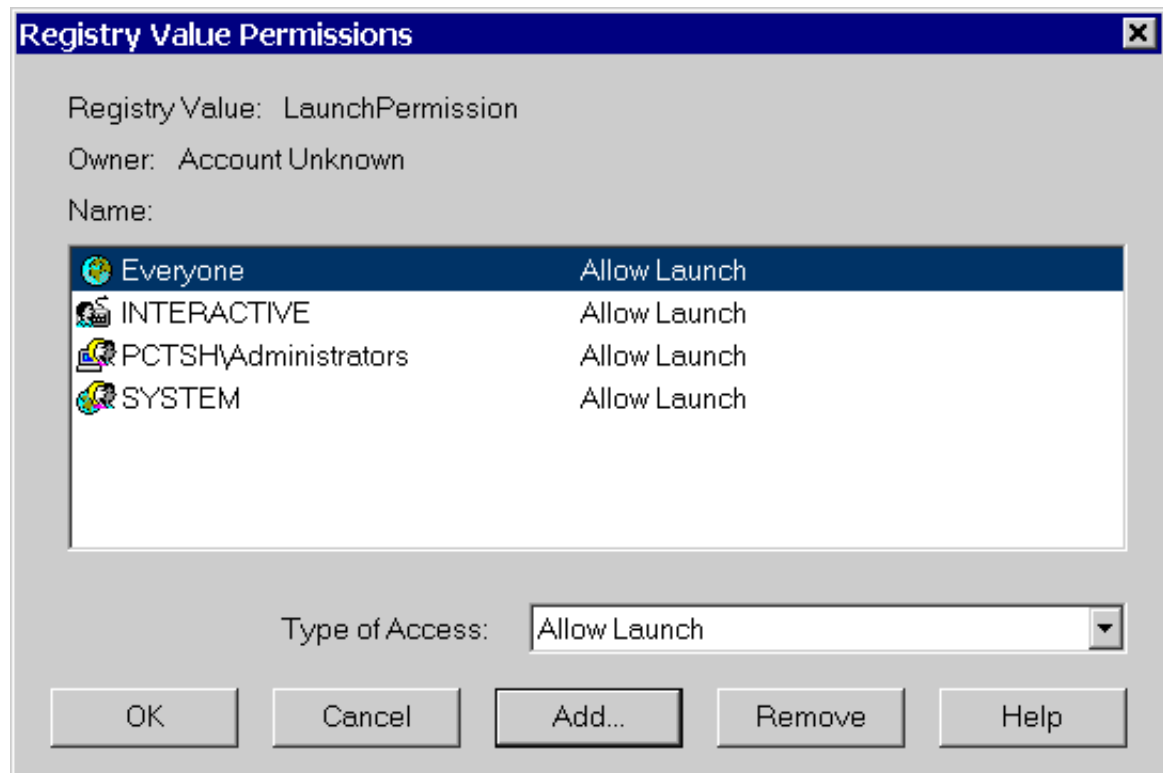
10. In the "Security" tab, make sure that the access permissions are set to "Use default access permissions".
11. Activate the "Use custom launch permissions" check box and choose "Edit" to modify the application-specific launch permissions.



The registry value "LaunchPermission" will contain at least the accounts SYSTEM and INTERACTIVE and the group Administrators.

12. Add the users and groups to be allowed to launch your NaturalX server.

In most cases, you will define a group of all users to whom you want to grant launch and enter this group here. In the example, the built-in group "Everyone" is entered. This grants launch to every user that is defined on the server machine. If the built-in account "Guest" is enabled in the User Manager, this setting grants launch to users not defined on the server machine (guests) as well.



13. In the "Identity" tab, define the account under which the NaturalX server will be launched.
- If you select "The launching user", a server process will be launched for each client. The server process will be launched under the account of the client user.
 - If you select "The interactive user", only one server process will be launched for all clients.

Note:

This is true only for classes that have been registered in Natural as "ExternalMultiple". If a class is registered as "ExternalSingle", a server process is created for each object of this class that is created.

The server process will be launched under the account of the user that is interactively logged in on the server machine. If no user is currently logged in on the server machine, this setting behaves like "The launching user".

- If you select "This user" and select a specific user account, only one server process will be launched for all clients.

Note:

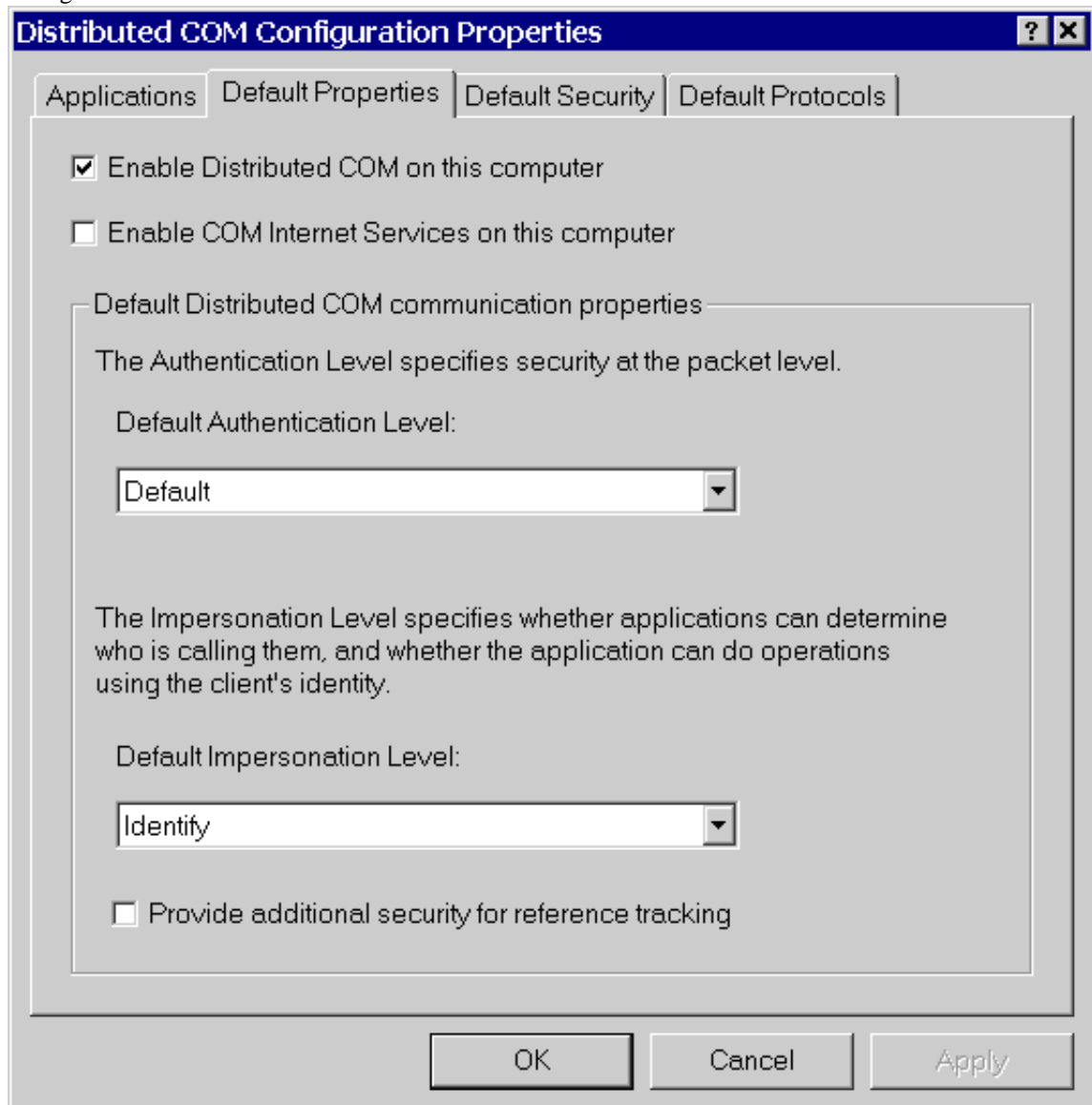
This is true only for classes that have been registered in Natural as "ExternalMultiple". If a class is registered as "ExternalSingle", a server process is created for each object of this class that is created.

The server process will be launched under the specified user account.

The screenshot shows a Windows-style dialog box titled "Natural Classes for Employees server Properties". It has a blue title bar with a question mark and a close button. Below the title bar are five tabs: "General", "Location", "Security", "Identity" (which is selected), and "Endpoints". The main area of the dialog contains the text "Which user account do you want to use to run this application?". There are four radio button options: "The interactive user", "The launching user" (which is selected), "This user:", and "The System Account (services only)". Under the "This user:" option, there are three text input fields labeled "User:", "Password:", and "Confirm Password:". To the right of the "User:" field is a "Browse..." button. At the bottom of the dialog are three buttons: "OK", "Cancel", and "Apply".

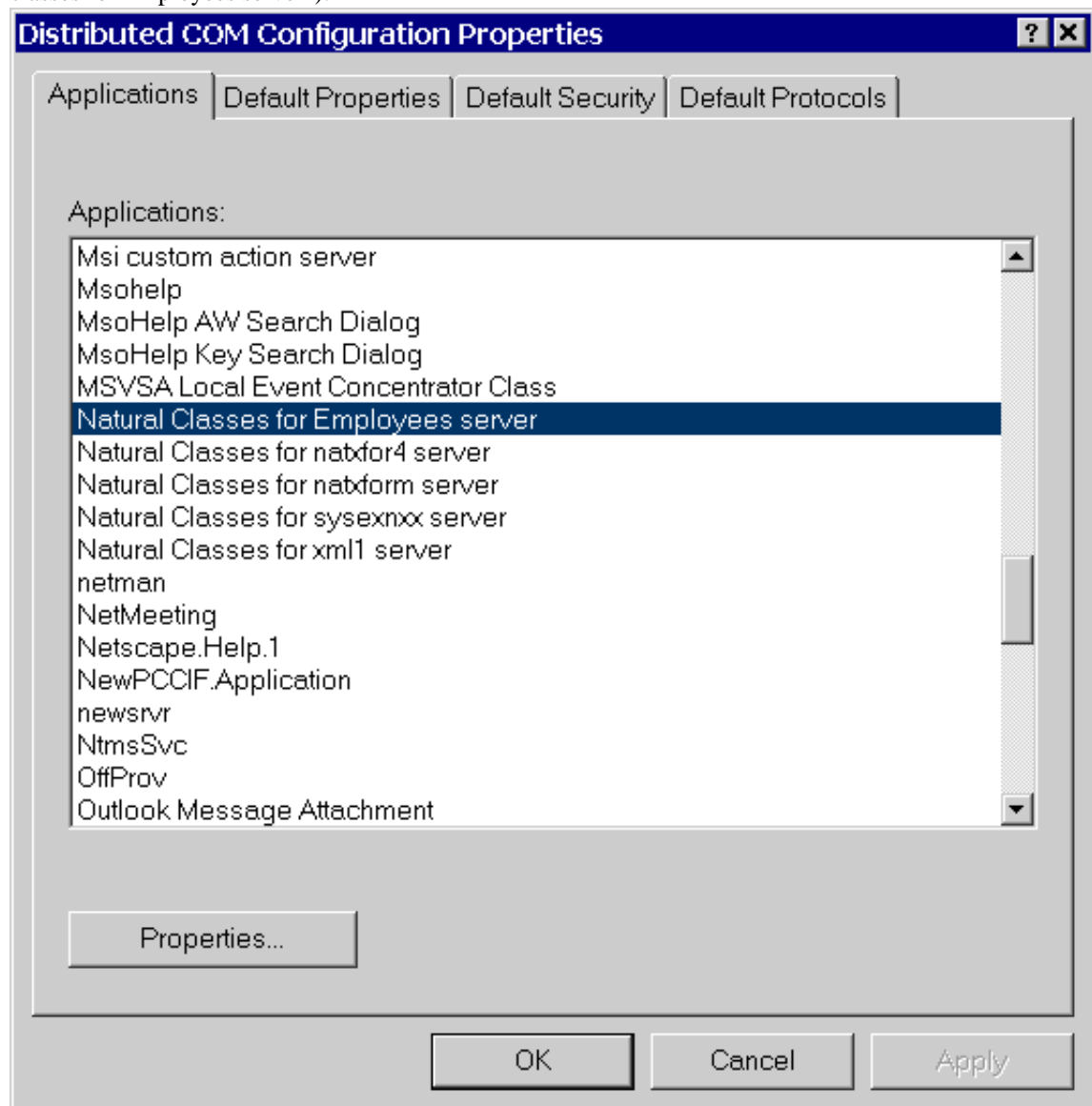
Configuring NaturalX Clients on Windows NT/2000

1. Invoke the "Distributed COM Configuration Properties" dialog box.
2. In the "Default Properties" tab, activate the checkbox "Enable Distributed COM on this computer".
3. Set "Default Authentication Level" to "Default" and "Default Impersonation Level" to "Identify".
This allows NaturalX servers to retrieve the client's user ID. Before executing a request, the server will then move the client's user ID into the Natural system variable *USER in order to let Natural Security checks run against this user ID.



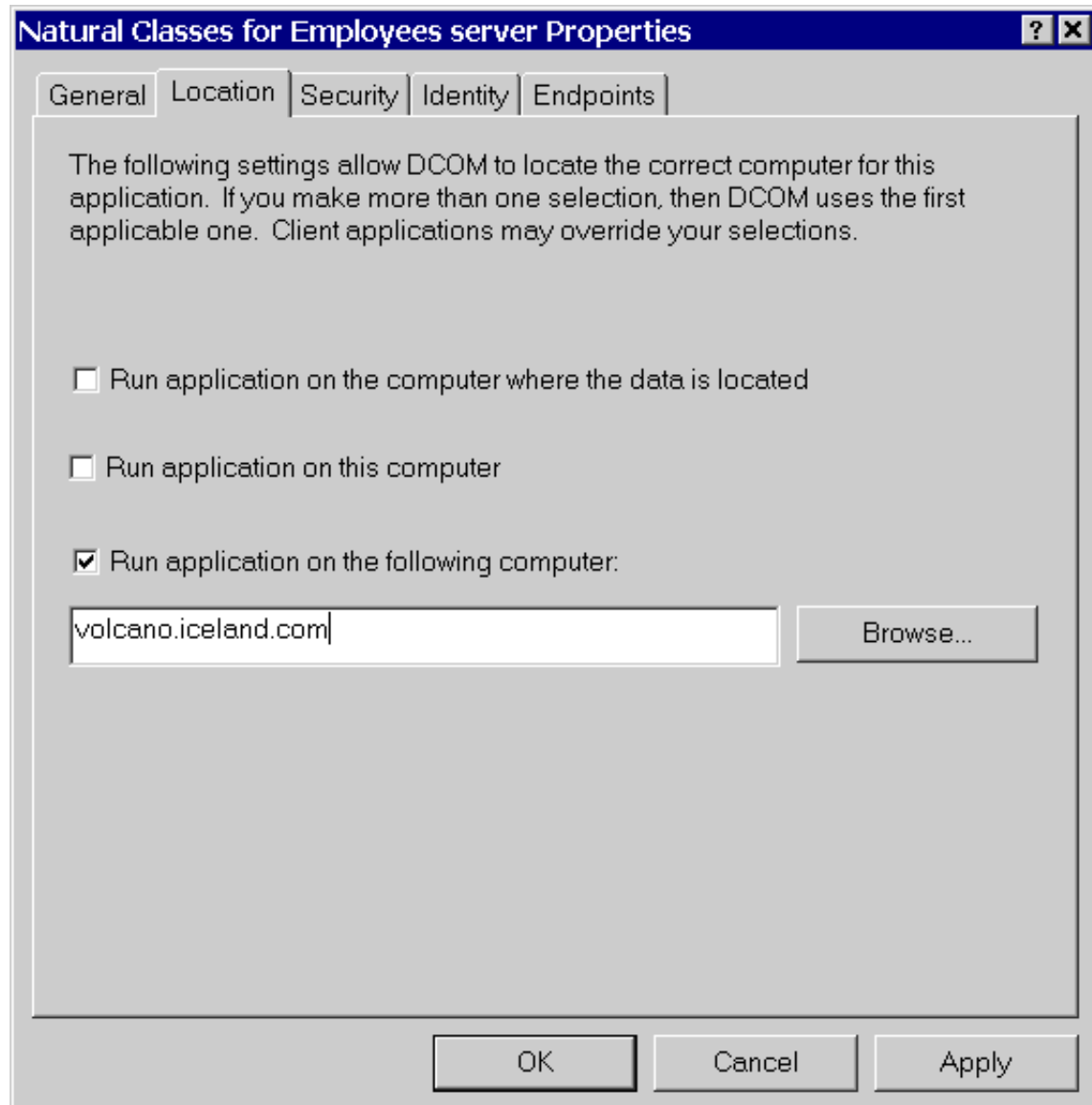
Now you can set up the configuration to access a specific NaturalX server.

4. In the "Applications" tab, locate your NaturalX server in the Applications list box (in the example "Natural classes for Employees server").



5. Select your server and choose "Properties".
6. In the "Location" tab, activate the checkbox "Run application on the following computer".

7. Enter the name of the remote machine on which the NaturalX server is installed.



DCOM Configuration on Windows 98

This section describes how to configure NaturalX applications on Windows 98 in a pure Windows 98 network, without a Windows NT domain server. All settings are applied with the tool DCOMCNFG.

Under Windows 98, DCOM is included. However, the tool DCOMCNFG might not be available in your installation. In this case, it must be installed separately. This product is freely available from Microsoft.

DCOM on Windows 98 differs from DCOM on Windows NT in the following ways:

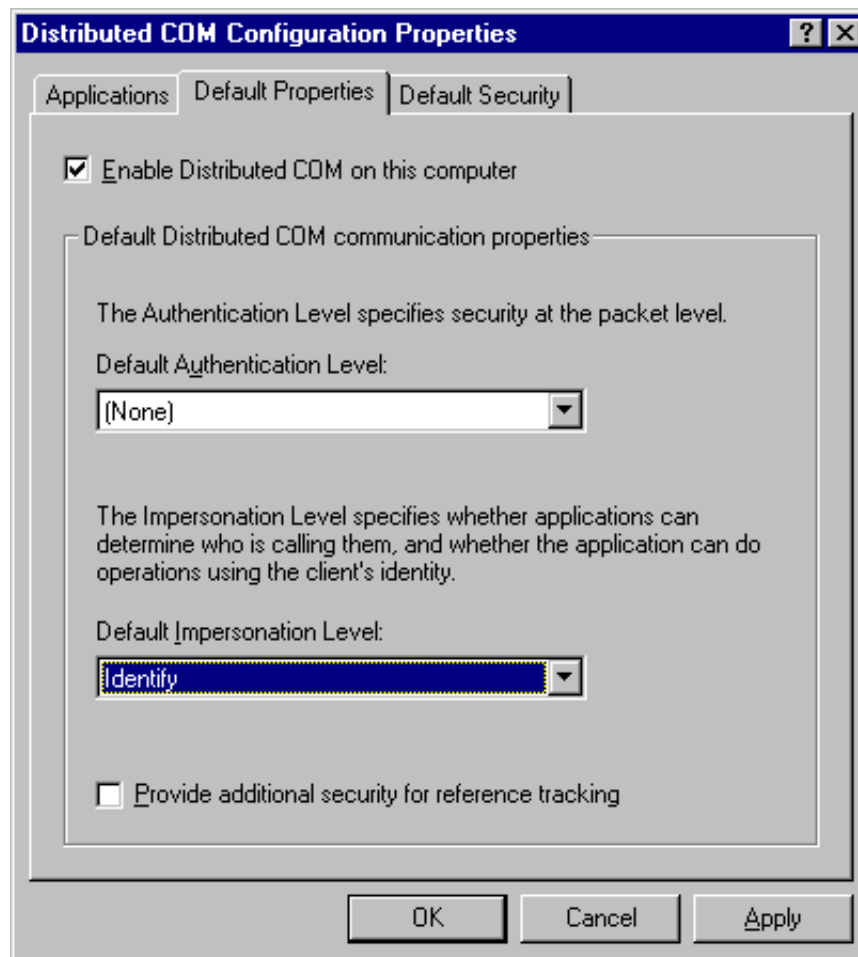
- Windows 98 lacks the security infrastructure available under Windows NT. In a pure Windows 98 network, no authenticated calls can be made. Therefore, NaturalX clients and servers must always run with Authentication Level "None".
- DCOM servers are not launched automatically. Therefore, NaturalX servers must be started manually in advance.
- Impersonation is not supported. Therefore, a NaturalX server always runs under the user account under which it was started manually.

This section covers the following topics:

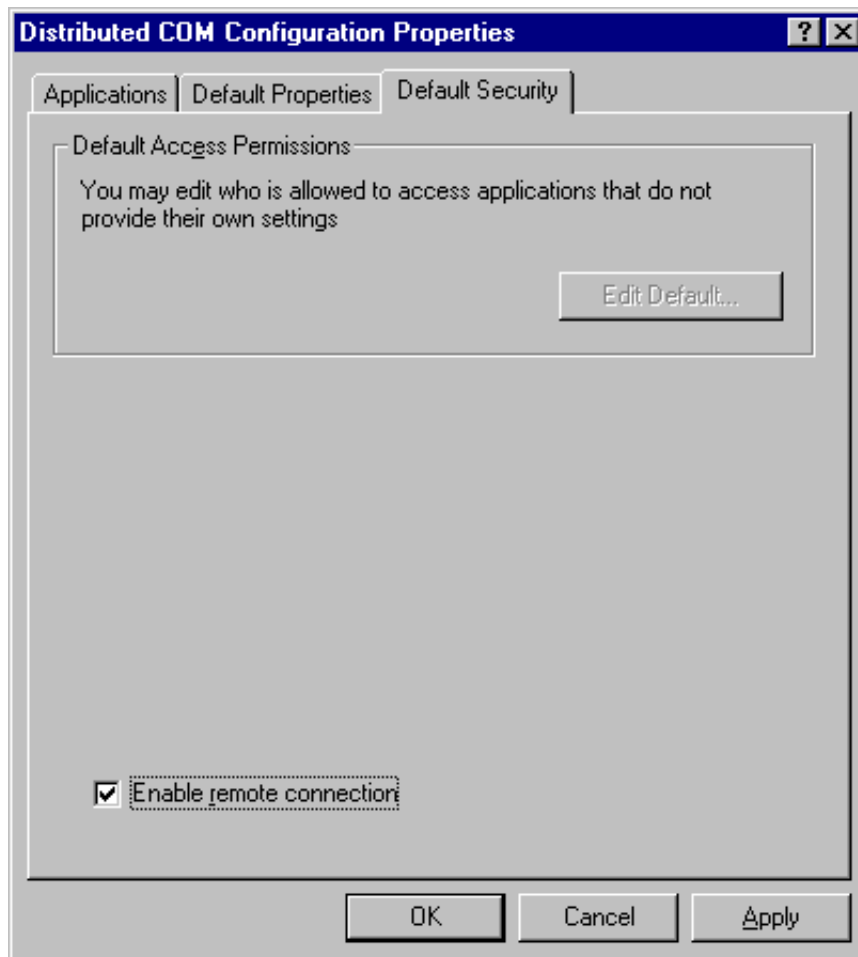
- Configuring NaturalX Servers on Windows 98
- Configuring NaturalX Clients on Windows 98

Configuring NaturalX Servers on Windows 98

1. Invoke the "Distributed COM Configuration Properties" dialog box.
2. In the "Default Properties" tab, activate the checkbox "Enable Distributed COM on this computer". Set "Default Authentication Level" to "None" and "Default Impersonation Level" to "Identify".



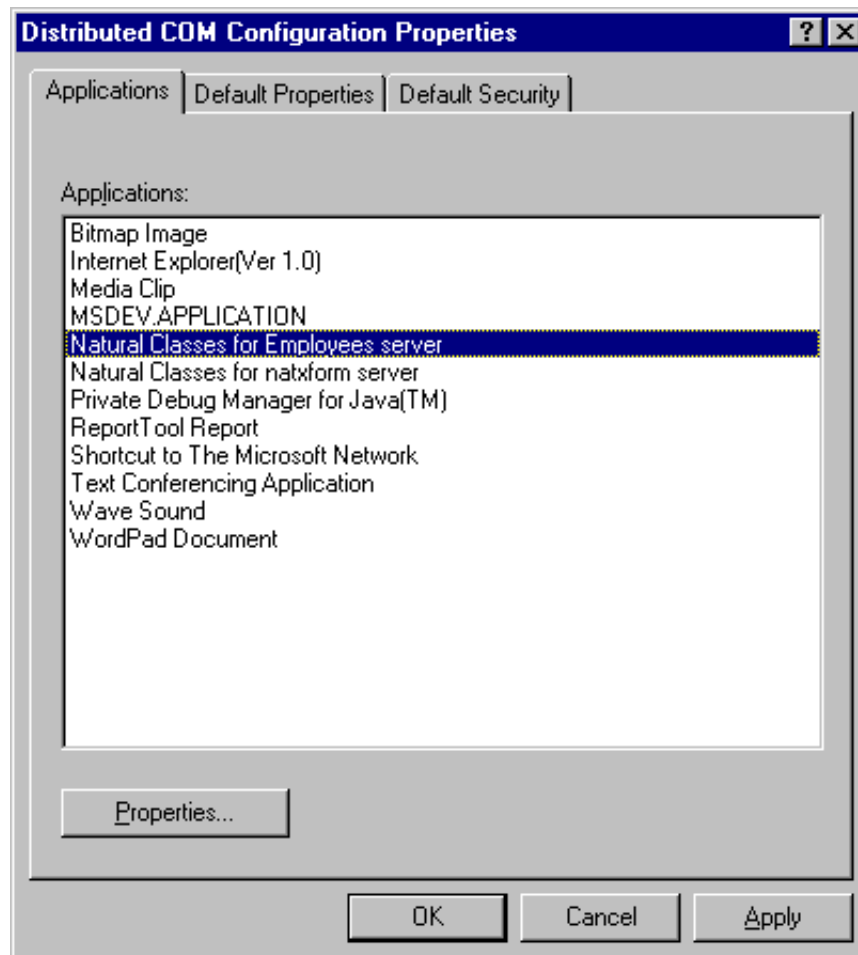
3. In the "Default Security" tab, activate the checkbox "Enable remote connection" to allow clients to establish remote DCOM connections to the server machine.



Now you can set up the configuration for a specific NaturalX server.

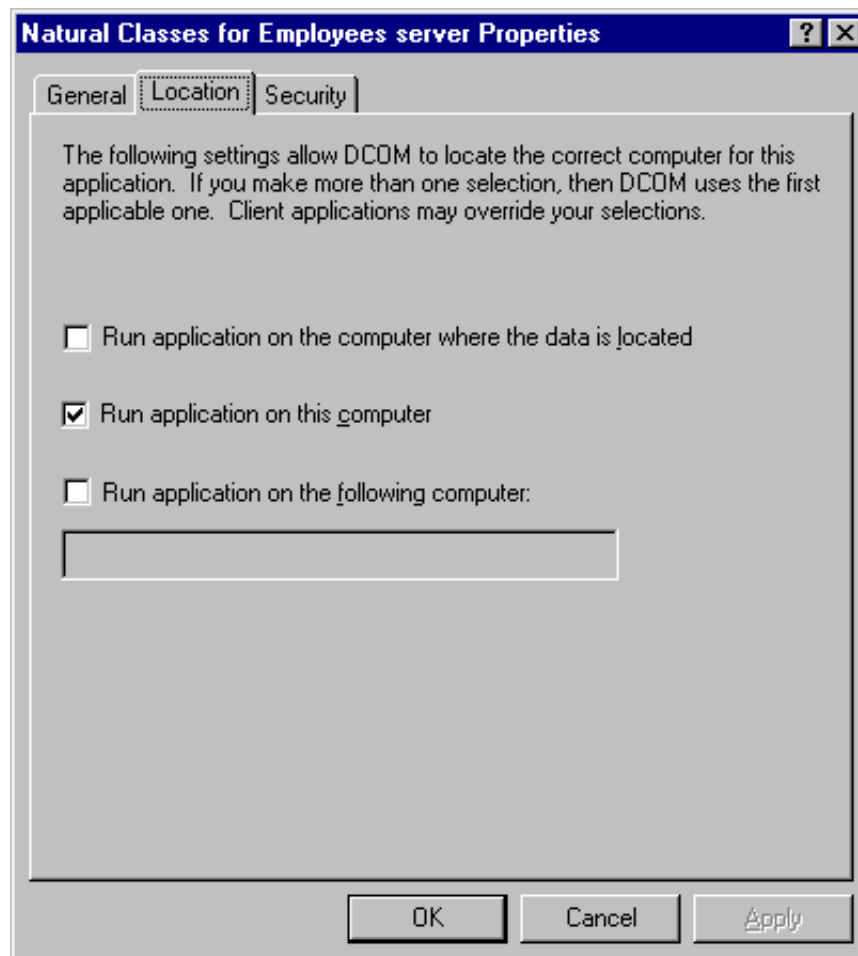
4. In the "Applications" tab, select your NaturalX server in the 'Applications' list box. Now you can set up the configuration for a specific NaturalX server.

A bug in DCOMCNFG means that under certain conditions it does not show the name of the server (in the example "Natural classes for Employees server"), but the name of one of the classes ("newemployee 1.0") instead.



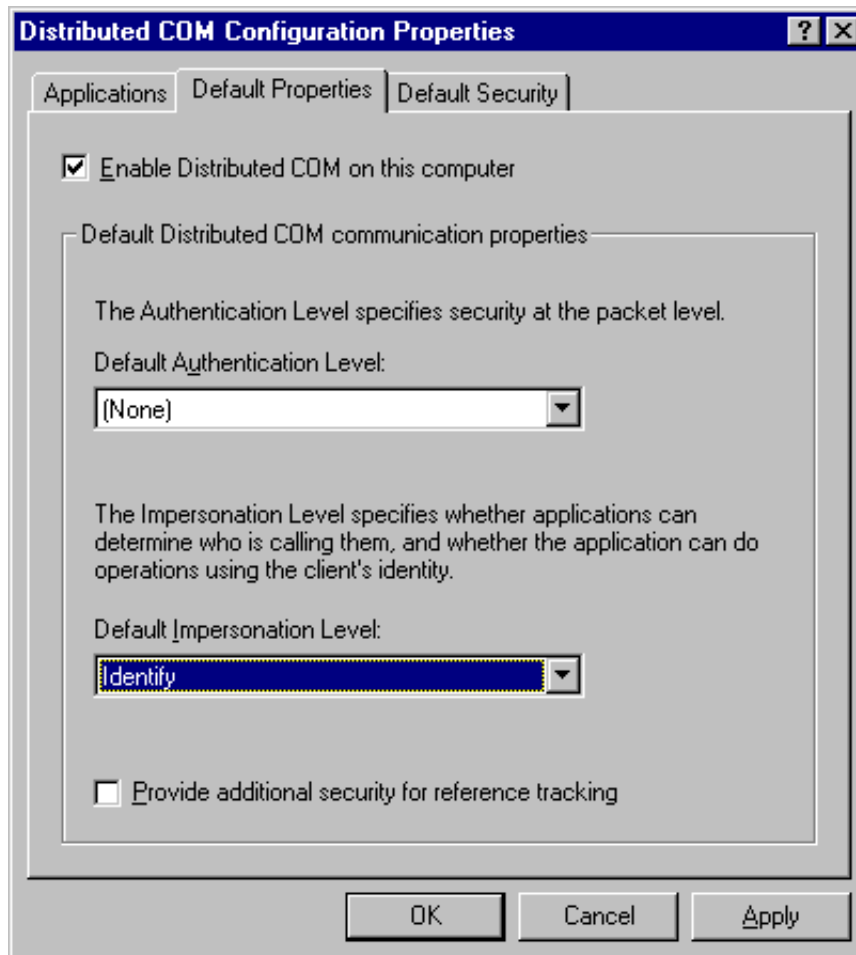
5. Choose "Properties".

6. In the "Location" tab, activate the checkbox "Run application on this computer".



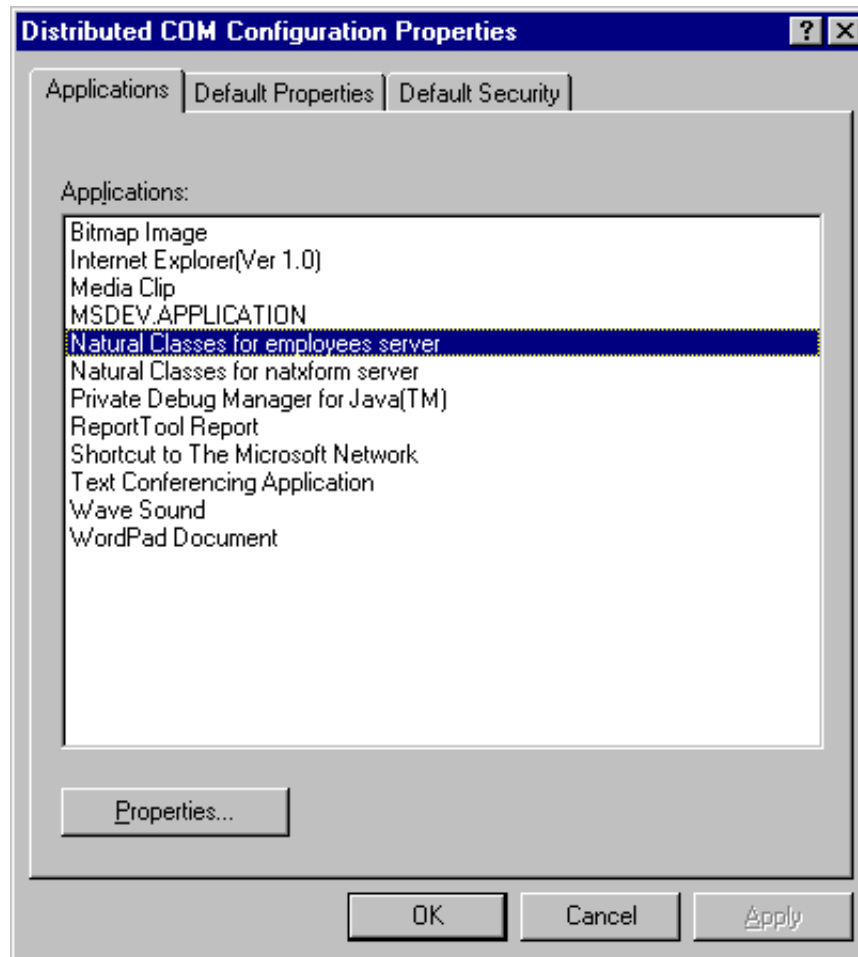
Configuring NaturalX Clients on Windows 98

1. Invoke the "Distributed COM Configuration Properties" dialog box.
2. In the "Default Properties" tab, activate the check box "Enable Distributed COM on this computer". Set "Default Authentication Level" to "None" and "Default Impersonation Level" to "Identify".



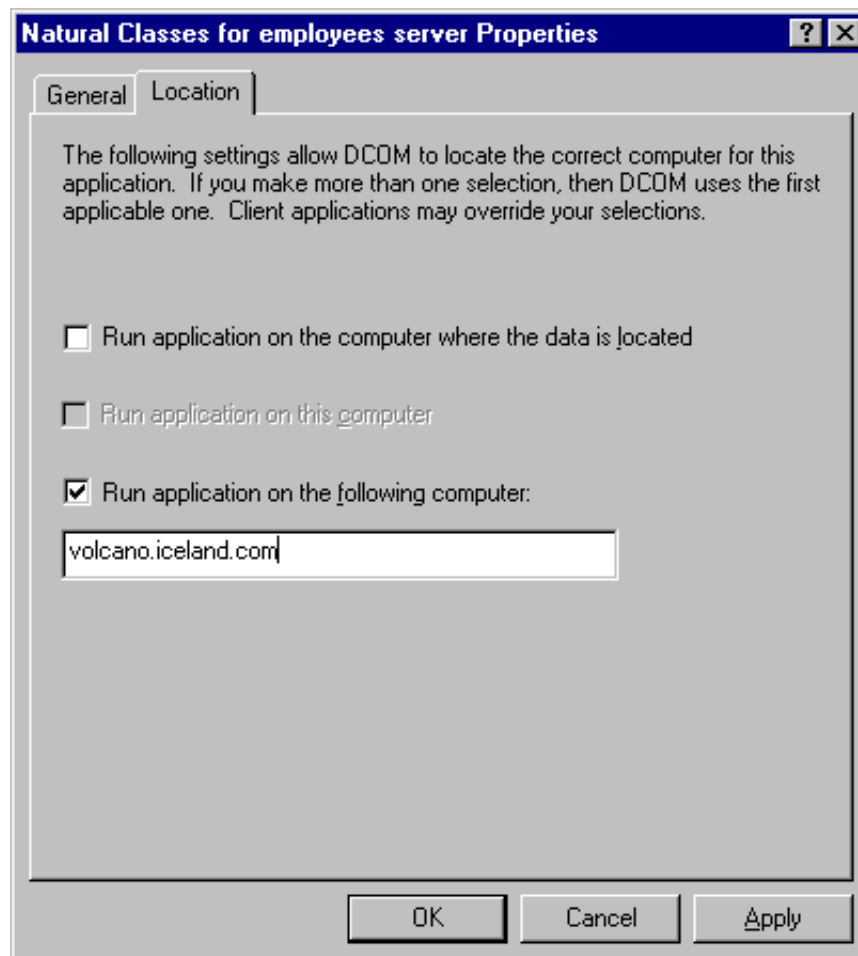
Now you can set up the configuration to access a specific NaturalX server.

3. In the "Applications" tab, select your NaturalX server from the list of DCOM applications.
A bug in DCOMCNFG, means that under certain conditions it does not show the name of the server (in the example "Natural classes for Employees server"), but the name of one of the classes ("newemployee 1.0") instead.



4. Choose "Properties".
5. In the "Location" tab, activate the checkbox "Run application on the following computer".

6. Enter the name of the remote machine on which the NaturalX server is installed.



DCOM Configuration on Windows 98 in a Windows NT Domain

This section describes how to configure NaturalX applications Windows 98, if a Windows NT domain server is available in the network. All settings are applied with the tool DCOMCNFG.

Under Windows 98, DCOM is included. However, the tool DCOMCNFG might not be available in your installation. In this case, it must be installed separately. This product is freely available from Microsoft.

DCOM Windows 98 differs from DCOM on Windows NT in the following ways:

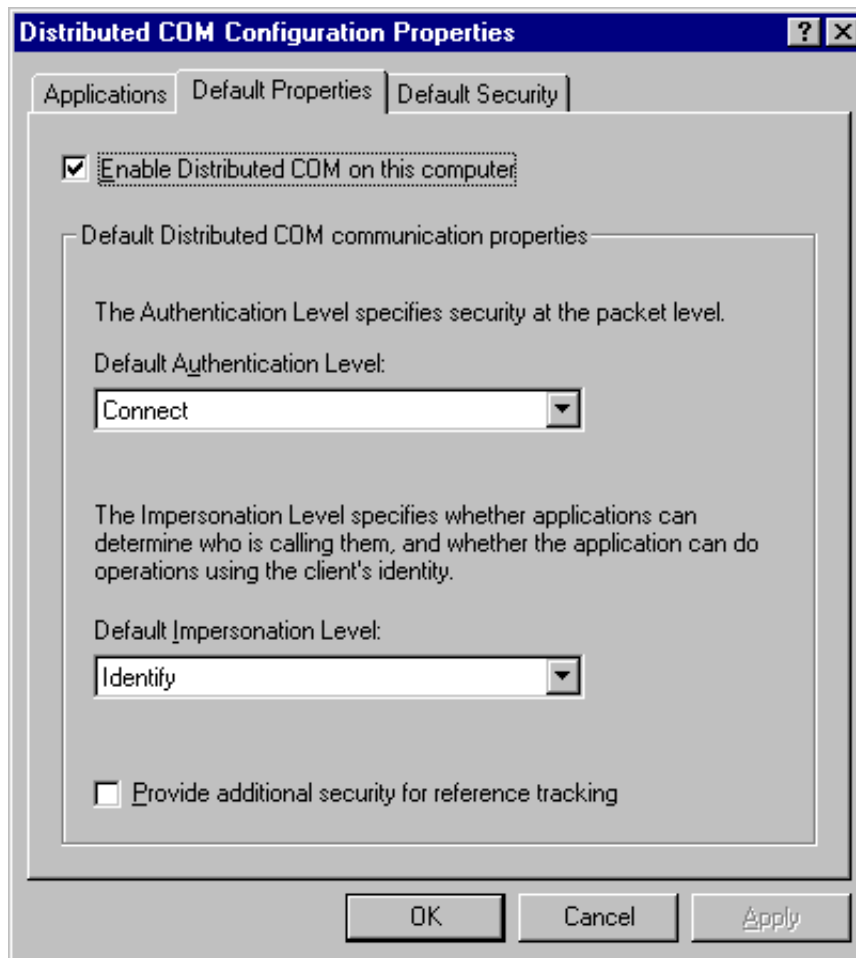
- Windows 98 lacks the security infrastructure available under Windows NT. Under Windows 98, authenticated DCOM calls can be made only with the help of a Windows NT domain server. Therefore, user-level access control must be activated on each Windows 98 machine that is to act as a server. This allows the use of a Windows NT domain server to authenticate DCOM requests. See the Windows 98 documentation on how to activate user-level access control.
- DCOM servers are not launched automatically. Therefore, NaturalX servers must be started manually in advance.
- Impersonation is not supported. Therefore, a NaturalX server runs always under the user account under which it was manually started.

This section covers the following topics:

- Configuring NaturalX Servers on Windows 98 in a Windows NT Domain
- Configuring NaturalX Clients on Windows 98 in a Windows NT Domain

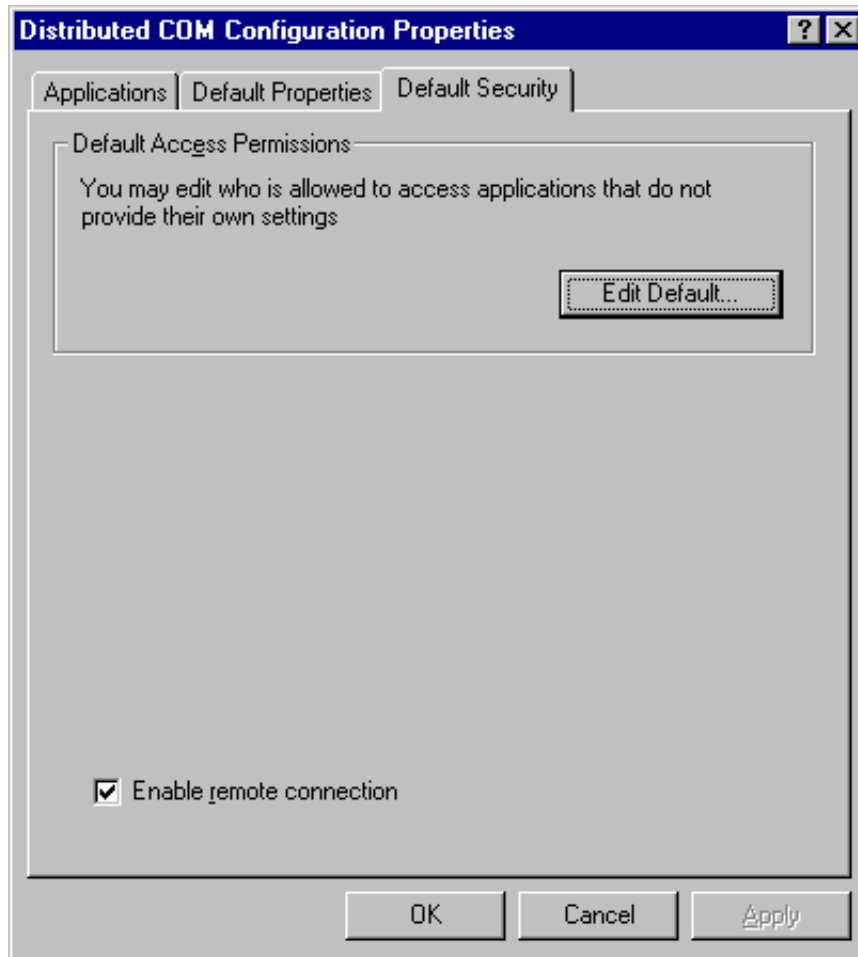
Configuring NaturalX Servers on Windows 98 in a Windows NT Domain

1. Invoke the "Distributed COM Configuration Properties" dialog box.
2. In the "Default Properties" tab, activate the checkbox "Enable Distributed COM on this computer".
3. Set "Default Authentication Level" to "Connect" and "Default Impersonation Level" to "Identify".
This allows NaturalX servers to retrieve the client's user ID. Before executing a request, the server will then move the client's user ID into the Natural system variable *USER in order to let Natural Security checks run against this user ID.

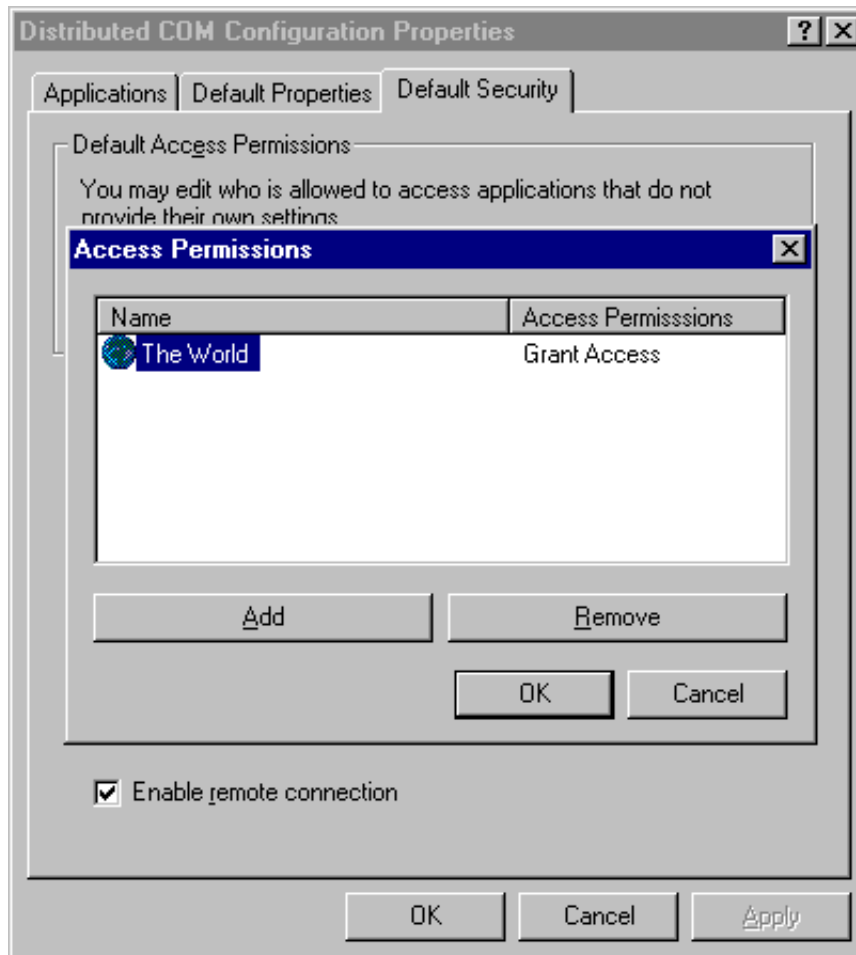


Now you can set up the default security configuration.

4. In the "Default Security" tab, activate the checkbox "Enable remote connection" to allow clients to establish remote DCOM connections to the server machine.



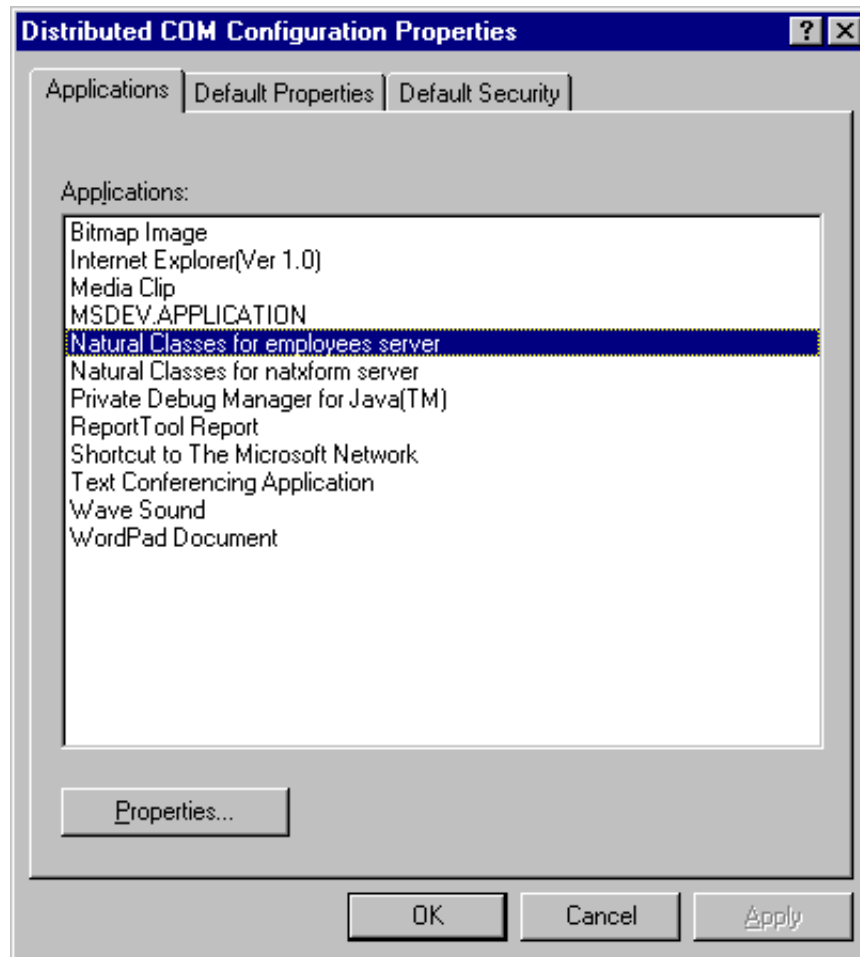
5. Choose "Edit default" to edit the default access permissions in the Access Permissions dialog.
The "DefaultAccessPermission" must contain the users and groups that shall be allowed to access NaturalX servers. In most cases you will define a group of all users to whom you want to grant access and enter this group here. In the example, the built-in group "The World" is entered. This grants access to every user that is defined in the domain. If the built-in account "Guest" is enabled in the User Manager on the domain server, this setting also grants access to users not defined in the domain (guests).



Now you can set up the configuration for a specific NaturalX server.

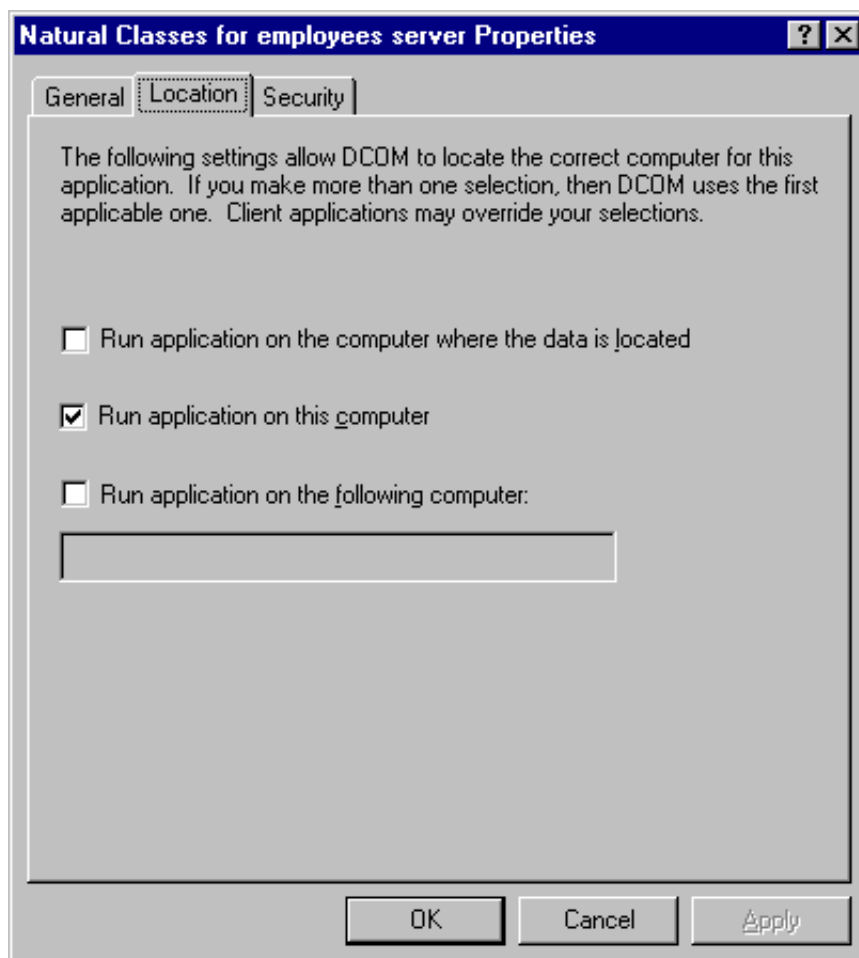
6. In the "Applications" tab, locate your NaturalIX server in the list.

A bug in DCOMCNFG means that under certain conditions it does not show the name of the server (in the example "Natural classes for employees server"), but the name of one of the classes ("newemployee 1.0") instead.

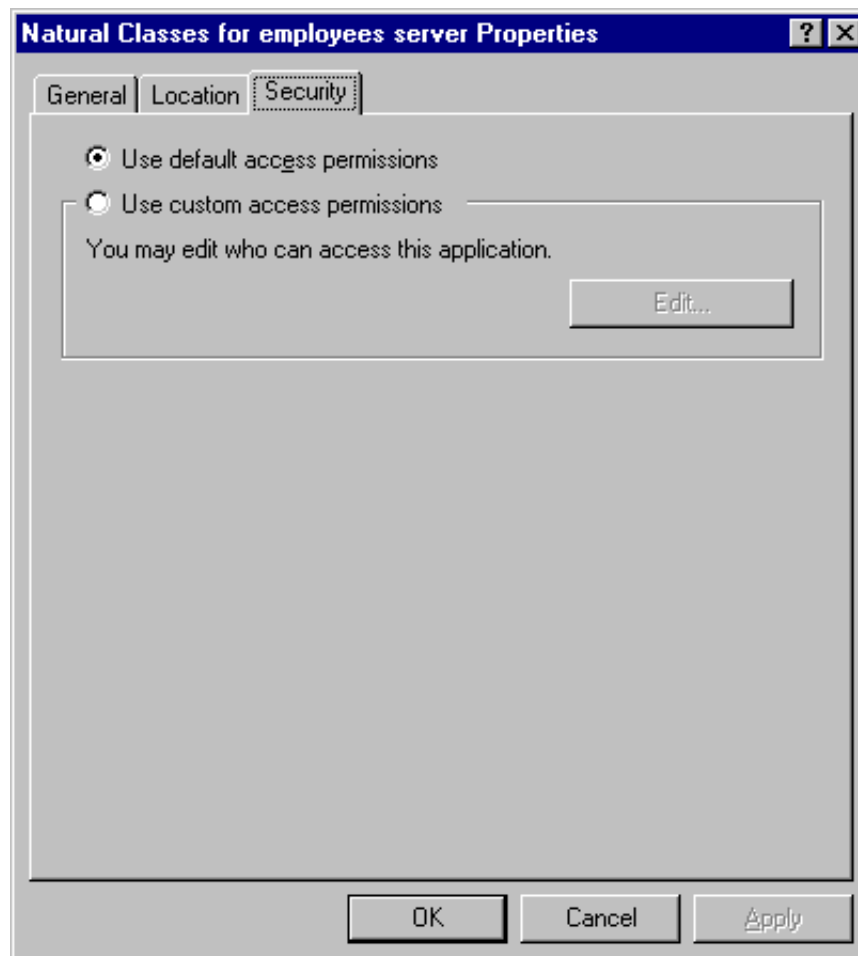


7. Select your server and choose "Properties".

8. In the "Location" tab, activate the checkbox "Run application on this computer".

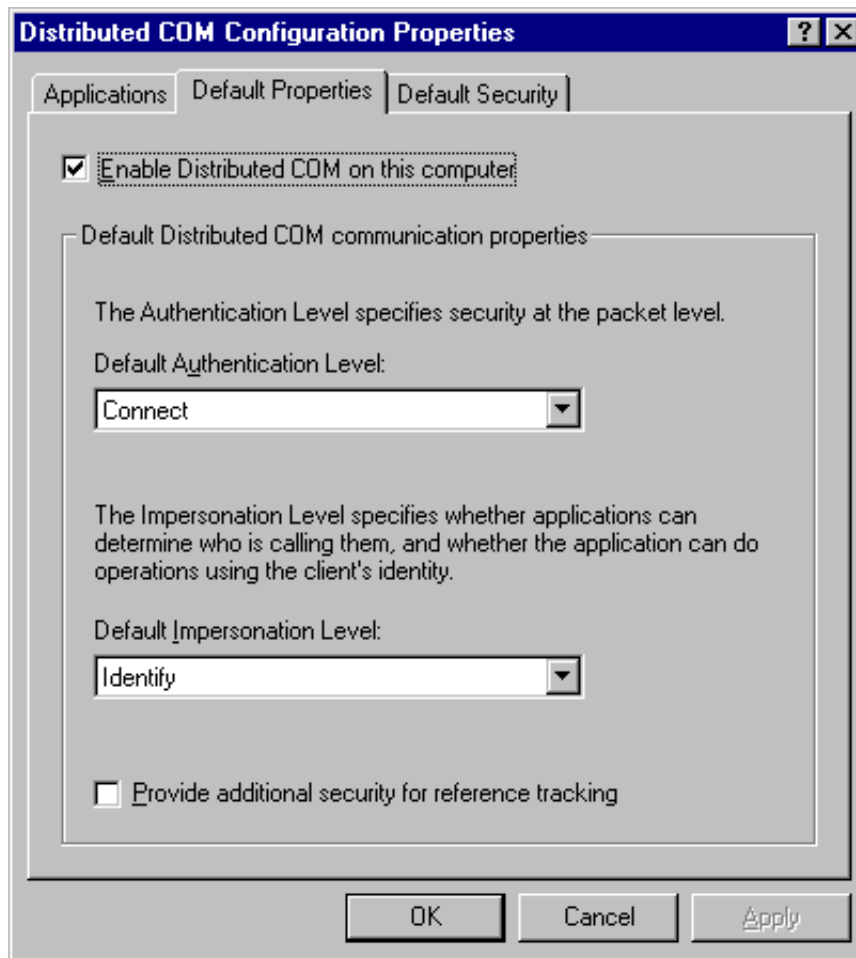


9. In the "Security" tab, make sure that the access permissions are set to "Use default access permissions".



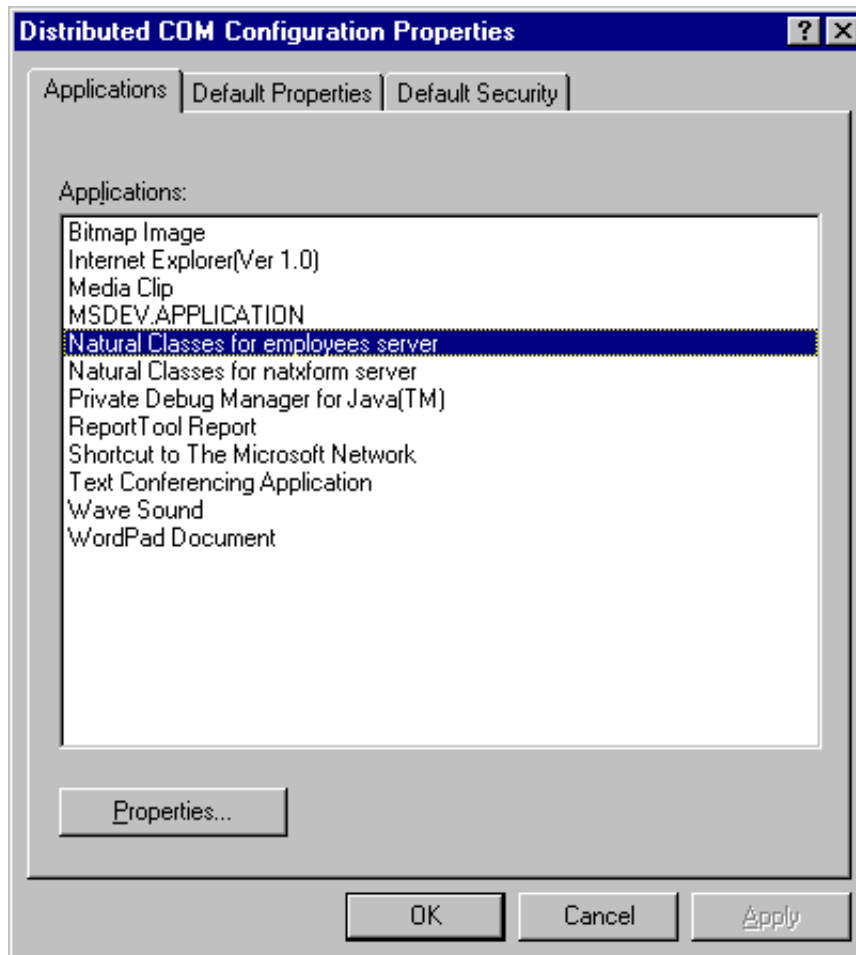
Configuring NaturalX Clients on Windows 98 in a Windows NT Domain

1. Invoke the "Distributed COM Configuration Properties" dialog box.
2. In the "Applications" tab, activate the checkbox "Enable Distributed COM on this computer".
3. Set "Default Authentication Level" to "Connect" and "Default Impersonation Level" to "Identify".
This allows NaturalX servers to retrieve the client's user ID. Before executing a request, the server will move the client's user ID into the Natural system variable *USER in order to let Natural Security checks run against this user ID.



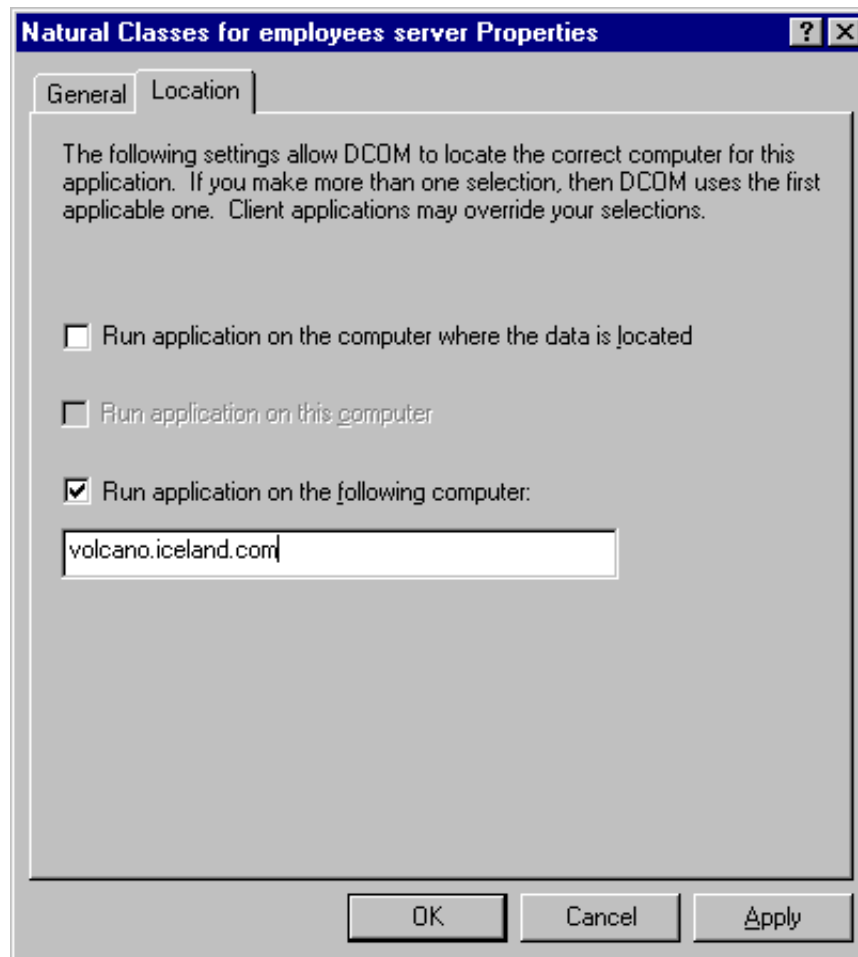
Now you can set up the configuration to access a specific NaturalX server

4. In the "Application" tab, locate your NaturalIX server in the list of DCOM applications.
A bug in DCOMCNFG means that under certain conditions it does not show the name of the server (in the example "Natural classes for employees server"), but the name of one of the classes ("newemployee 1.0") instead.



5. Select your server and choose "Properties....".

The "Natural Classes for employees server Properties" dialog box appears.



6. In the "Location" tab, activate the checkbox "Run application on the following computer".
7. Enter the name of the remote machine on which the NaturalX server is installed.

DCOM Configuration on UNIX with EntireX

This section describes how to configure NaturalX applications on UNIX under EntireX DCOM.

EntireX DCOM contains a command line utility DCOMCNFG that provides functions to configure DCOM applications, similar to DCOMCNFG.EXE on Windows NT. For detailed documentation of this utility and for more detailed information on DCOM Security on UNIX, see your EntireX DCOM documentation.

You can only use DCOMCNFG to give access and launch permissions to users which have already been defined to EntireX DCOM. EntireX DCOM provides the following two methods of user authentication:

- Using a local password file
- Using a Windows NT domain server

For further information, see your EntireX DCOM documentation.

This section covers the following topics:

- Configuring NaturalX Servers on UNIX
- Configuring NaturalX Clients on UNIX

Configuring NaturalX Servers on UNIX

1. Enable DCOM on the server machine as in the following example:

```
dcomcnfg EnabledDCOM=Y
dcomcnfg LegacyAuthenticationLevel=Default
dcomcnfg LegacyImpersonationLevel=Identify
```

Setting "LegacyAuthenticationLevel" to "Default" and "LegacyImpersonationLevel" to "Identify" allows NaturalX servers to retrieve the client's user ID. Before executing a request, the server will move the client's user ID into the Natural system variable *USER in order to let Natural Security checks run against this user ID.

Now you can set up the "Default Security" configuration.

2. Set the default access permissions. This defines which users and groups can access NaturalX servers. In most cases you will define a group of all users to whom you want to grant access and enter this group here. In the example, the built-in group "Everyone" is entered. This grants access to every user that is defined on the server machine.

```
dcomcnfg DefaultAccessPermission=Everyone
```

Set the default launch permissions. This defines which users and groups can launch NaturalX servers. The registry value "DefaultLaunchPermission" must contain at least the account "System"

```
dcomcnfg DefaultLaunchPermission=System
```

3. Run the command **dcomcnfg** without parameters.
This lists among other information all defined AppIDs.
4. Select the AppID of your NaturalX server.
5. Specify that the NaturalX server will run on this machine (the server machine).

```
dcomcnfg "{088726A0-4718-11D2-BF75-080020789C1E}" RemoteServerName="This computer"
```

Note:

This command must be entered in one line.

6. Define who shall be allowed to launch your NaturalX server.

In most cases you will create a group containing all users to whom you want to grant launch and enter this group here. In the following example the built-in group "Everyone" is entered. This allows launch to every user defined on the server machine:

```
dcomcnfg " {088726A0-4718-11D2-BF75-080020789C1E} "LaunchPermission=Everyone
```

7. Define the account under which the NaturalX server shall be launched.

If you select "Launching User", an own server process will be launched for each different client. The server process will be launched under the account of the client user.

```
dcomcnfg " {088726A0-4718-11D2-BF75-080020789C1E} " RunAs="Launching User"
```

If you select a specific user account, only one server process will be launched for all clients. Note that this is true only for classes that have been registered in Natural as "ExternalMultiple". If a class is registered as "ExternalSingle", an own server process is created anyway for each object that is created of this class. The server process will be launched under the specified user account.

```
dcomcnfg " {088726A0-4718-11D2-BF75-080020789C1E} " RunAs=Scully
```

OS/390 UNIX Services Only

If you select a specific user account, only one server process will be launched for all clients. This is the recommended setting for NaturalX under OS/390. NaturalX separates classes with the activation policies "ExternalSingle" and "ExternalMultiple" in different Natural sessions.

Configuring NaturalX Clients on UNIX

1. Enable DCOM on the server machine.

Setting "LegacyAuthenticationLevel" to "Default" and "LegacyImpersonationLevel" to "Identify" allows NaturalX servers to retrieve the client's user ID. Before executing a request, the server will then move the client's user ID into the Natural system variable *USER in order to let Natural Security checks run against this user ID.

```
dcomcnfg EnableDCOM=Y
dcomcnfg LegacyAuthenticationLevel=Default
dcomcnfg LegacyImpersonationLevel=Identify
```

Now you can set up the application-specific configuration.

2. Run the command **dcomcnfg** without parameters.
This lists among other information all defined AppIDs.
3. Select the AppID of your NaturalX server.
4. Enter the name of the remote machine on which the NaturalX server is installed.

```
dcomcnfg "{088726A0-4718-11D2-BF75-080020789C1E}" RemoteServerName="volcano.iceland.com"
```

Note:

This command must be entered in one line.

DCOM Configuration on OS/390

See the section DCOM Configuration on UNIX

NaturalX System Registry Entries

This section covers the following topics:

- Registry Entries for Servers
 - Registry Entries for Clients
-

Registry Entries for Servers

The following tables show a summary of the keys and values that are added in the system registry of the server when a new class is registered.

The column "parent key" shows under which key the new key is created. The key which is added is listed in the column "subkey", and the columns "value name" and "value" show the value of the new entry.

Note:

<class_name> and <class_ID> are the name and the class GUID of the class respectively. They are defined in the DEFINE CLASS statement of the class module.

- Keys Needed by DCOM
- Keys Needed by Natural

Keys Needed by DCOM

parent key (HKEY_CLASSES_ROOT...)	subkey	value name	value
...	<ProgID> (<class_name>.1)	-	<class_name> "1.0"
... \<ProgID>	CLSID	-	<class_GUID>
...	<VersIdProgID> (<class_name>)	-	<class name> "1.0"
... \<VersIdProgID>	CLSID	-	<class GUID>
... \AppId	<APPID>	-	"Natural classes for " <server_ID> "server"
... \CLSID	<CLSID>	-	<class_name> "1.0"
... \CLSID	<CLSID>	AppId	<GUID for server>
... \CLSID \<CLSID>	LocalServer32	-	<Natural path>
... \CLSID \<CLSID>	ProgID	-	<ProgID>
... \CLSID \<CLSID>	TypeLib	-	<GUID for type library>
... \CLSID \<CLSID>	Version	-	"1.0"
... \CLSID \<CLSID>	VersionIndependentProgID	-	<VersIDProgID>
... \CLSID \<CLSID> (applies for Version 4.1.2 and all subsequent releases)	Programmable	-	-
... \TypeLib	<TLID>	-	-
... \TypeLib\<TLID>	1.0 <version>	-	"Natural" <class_name> "class"
... \TypeLib\<TLID>\1.0	0 (langcode)	-	-
... \TypeLib\<TLID>\1.0\0	win32 (platform)		<type library path>
For every interface:			
... \Interface	<IID>	-	<interface name>
... \Interface\<IID>	ProxyStubClsid32	-	<GUID of proxy dll for IDispatch>
... \Interface\<IID>	BaseInterface	-	<GUID of IDispatch>

Keys Needed by Natural

parent key (HKEY_LOCAL_MACHINE\ SOFTWARE\SoftwareAG\ Natural\Servers...)	subkey	value name	value
...	<server_ID>	AppId	<GUID for server>
... \<server_ID>\	CLSID	-	-
... \<server_ID>\CLSID	<CLSID> (<class_ID>)	NatMember	<Natural class module name>
... \<server_ID>\CLSID	<CLSID>	NatLibrary	<Natural library of class module>
... \<server_ID>\CLSID	<CLSID>	NatContext	"ExternalSingle" or "InternalMultiple" or "ExternalMultiple" (see Activation Policies)

Registry Entries for Clients

The following table shows the keys which are added in the client system registry when the client registration file is executed:

parent key (HKEY_CLASSES_ROOT...)	subkey	value name	value
...	<ProgID> (<class_name>.1)	-	<class_name> "1.0"
... \<ProgID>	CLSID	-	<class GUID>
...	<VersIdProgID> (<class_name>)	-	<class_name> "1.0"
... \<VersIdProgID>	CLSID	-	<class GUID>
... \<VersIdProgID>	CurVer	-	<ProgID>
... \AppId	<APPID>	-	"Natural classes for server" <server_ ID> "server"
... \AppId	<APPID>	RemoteServerName	has to be entered by user
... \CLSID	<CLSID>	-	<class_name> "1.0"
... \CLSID	<CLSID>	AppId	<GUID for server>
... \CLSID \<CLSID>	ProgID	-	<ProgID>
... \CLSID \<CLSID>	Version	-	"1.0"
... \CLSID \<CLSID>	VersionIndependent ProgID	-	<VersProgID>
... \CLSID \<CLSID> (applies for Version 4.1.2 and all subsequent releases)	Programmable	-	-
For every interface:			
... \Interface	<IID>	-	<interface name>
... \Interface\<IID>	ProxyStubClsid32	-	<GUID of proxy dll for IDispatch >
... \Interface\<IID>	BaseInterface	-	<GUID of IDis- patch>

Using Statements and Commands in a NaturalX Server Environment

The behaviour of some Natural statements and Natural system commands changes in a server environment. This section covers the following topics:

- Natural Statements
 - Natural System Commands
-

Natural Statements

This section covers the following statements:

- DISPLAY, INPUT, PRINT, REINPUT and WRITE Statements
- WRITE WORK FILE and READ WORK FILE Statements
- STOP and TERMINATE Statements

DISPLAY, INPUT, PRINT, REINPUT and WRITE Statements

- Output to a screen (output to Report 0) is not appropriate in a server environment, and in some cases is not possible. Therefore, in the case of an interactive I/O in the server environment, the error NAT0723 is returned to the client. Redirecting the I/O by using the MAINPR parameter is, of course, possible and is fully supported.
- When output is written to a report by a method, the report is opened at the start of the method and closed at the end. The report is not kept open between method calls to avoid interference between clients.

OS/390

Open/close processing is controlled by the OPEN/CLOSE option of the corresponding PRINTER/WORK parameter definition. This means that files may be kept open between method calls. Shared print/work files are closed at server termination only.

Note:

Print/work files whose name starts with the letters 'CM' are shared.

WRITE WORK FILE and READ WORK FILE Statements

Windows 98/NT/2000 and UNIX

- When you access a work file in a method, the file is opened at the start of the method and closed at the end. The file is not kept open between method calls to avoid interference between clients.

OS/390

Shared Files

- Shared files are opened at first access and remain open until the entire server process terminates.

Exclusive Files

- CLOSE=FIN
The work file remains open as long as the Natural session hosts objects. When the last object is released, the session terminates and closes the files.
- CLOSE=USR
The work file remains open until it is either closed explicitly by a CLOSE WORKFILE statement or until the session terminates.
- CLOSE=CMD
The work file is closed after each method execution.

STOP and TERMINATE Statements

- The behaviour of the TERMINATE statement matches that of the STOP statement. Processing of return values is not supported.
- The STOP and TERMINATE statements behave in the same way as the ESCAPE ROUTINE statement during method execution. Method execution is terminated immediately without producing any return value.

Natural System Commands

Only the following Natural commands are allowed in the server environment:

- CLEAR
- EXECUTE
- LOGOFF
- LOGON
- READ
- RETURN
- RUN
- SETUP

All other commands will be rejected with the error NAT0082.

NaturalX Glossary

This glossary explains the terminology used in this documentation.

Activation Policies

An activation policy is an attribute of a NaturalX class which defines whether it runs within its own exclusive Natural session or whether it may share a Natural session with other classes.

NaturalX combines the different options supported by DCOM in the form of the following three activation policies:

- *ExternalMultiple*
- *ExternalSingle*
- *InternalMultiple*

The activation policy of a class can be set as part of the REGISTER command, in the DEFINE CLASS statement or with the profile parameter ACTPOLICY=*activation-policy* (for OS/390, DCOM=(ACTPOL=*activation-policy*)).

AppID

In the system registry, each application is represented by an AppID. The AppID is a globally unique ID which can be found under the registry key HKEY_CLASSES_ROOT\AppID. DCOM uses the AppID to group classes to applications. Also, for example, security settings are defined on the basis of the AppID. Natural creates for each server ID one AppID in the registry.

For further information, see the section Server ID.

Class

Following the object-based programming approach, NaturalX classes encapsulate data structures (objects) with corresponding functionality (methods).

The internal structure of the objects of a class object is defined with a data area (object data area). The methods of a class are implemented as subprograms.

NaturalX classes can be made known to DCOM using the Natural command REGISTER, after which they are accessible in a network.

Classes can be *internal*, *external*, or *local*.

For further information on classes, see the sections Programming Techniques, Defining Classes, Using Classes and Objects, and Internal, External and Local Classes.

Class GUID

If a Natural class is to be registered as a DCOM class, a globally unique ID (GUID) must be defined for the class, to make sure it can be unambiguously identified in a network. In Natural, a GUID is assigned to a class in the ID clause of the DEFINE CLASS statement. A GUID is represented by an alphanumeric constant which can be generated in the data area editor.

Class Name

The name defined in the *class-name* operand in the DEFINE CLASS statement. This name is used in the CREATE OBJECT statement to create objects of that class.

Class Module Name

The name of the Natural module in which a Natural class is defined.

COM

Component Object Model. Microsoft specification for binary component API. Standardizes communication between components on a binary level. This component software model specifies how software components interact, irrespective of the programming language in which they were implemented.

For further information, see the Microsoft COM specification.

COM Class

A class that makes its methods and properties available to clients through interfaces that comply with the COM specification.

For further information, see the section COM.

DCOM

Distributed Component Object Model. DCOM extends COM to a distributed component software model which specifies how software components interact in a distributed environment.

With EntireX DCOM, Software AG has also made the DCOM technology available on UNIX and mainframe platforms.

External Class

A Natural class can be a local, an internal or an external class. This depends on the way the class was registered. An external class is a class that has been registered as a DCOM class with the REGISTER command option *ExternalSingle (ES)* or *ExternalMultiple (EM)*. Objects of external classes can be created and accessed by other processes. (With Natural on Windows 98/NT/2000 and UNIX, objects of external classes can also be created in the client process, provided the client process is at the same time a server process for the class.)

For further information, see the sections Local Class and Internal Class.

GUID

A GUID (globally unique identifier) is a constant that is guaranteed to be unique worldwide in the COM/DCOM model. It is used to unambiguously identify classes and their interfaces in a network. If a Natural class is to be registered as DCOM class, a GUID must be assigned to the class and to each of its interfaces. In Natural, a GUID is represented by an alphanumeric constant which can be generated in the data area editor.

For further information, see the section Globally Unique Identifiers (GUIDs).

HFS

Hierarchical File System. - UNIX file system available with OS/390 UNIX Services.

Instance

In the object-oriented programming model, data structures and functions (so called *methods*) are packaged together in objects. Each object belongs to a class, which describes the internal structure of the object and its interfaces, properties and methods. If an object belongs to a certain class, it is also called an instance of that class.

Interface

Interfaces are used by classes to provide clients with services. An interface is a collection of methods and properties. A client accesses these services by creating an object of the class and using the methods and properties of its interfaces.

You define an interface as follows:

- Define the INTERFACE clause to specify an interface name.
- Define the properties of the interface with PROPERTY definitions.
- Define the methods of the interface with METHOD definitions.

Interface GUID

If a Natural class is to be registered as a DCOM class, a globally unique ID (GUID) must be defined for each of its interfaces, to make sure the interfaces can be unambiguously identified in a network. The GUID is assigned to an interface in the ID clause of the INTERFACE statement. In Natural, a GUID is represented by an alphanumeric constant, which can be generated in the Data Area Editor.

Interface Inheritance

Interface inheritance means giving different classes the same interfaces, but implementing the interfaces differently in the different classes. This makes it possible to write client programs that only rely on these interfaces and are able to work with any class that has these interfaces.

Internal Class

A Natural class can be a local, an internal or an external class. This depends on the way the class was registered. An internal class is a class that has been registered as a DCOM class with the REGISTER command option *InternalMultiple (IM)*. Objects of internal classes can not be created by other processes, but they can be accessed by other processes. This requires that the object has been passed to the client process for example as return value of a method.

For further information, see the sections Local Class and External Class.

Local Class

A Natural class can be a local, an internal or an external class. This depends on the way the class was registered. A local class is a class that has not been registered as a DCOM class. Therefore, objects of local classes can neither be created nor accessed by other processes, but only by programs in the current Natural session.

For further information, see the sections Internal Class and External Class.

Method

A method is a function that an object/instance of a class can perform when requested by a client.

NaturalX Client

A NaturalX client is a process which creates or accesses NaturalX objects.

NaturalX Server

A NaturalX server is a process which manages one (Windows 98/NT/2000 and UNIX) or multiple (OS/390) Natural sessions. The Natural sessions managed by a NaturalX server are used to host COM objects.

Natural Session

A Natural session is the user-dependent Natural runtime context required for the Natural runtime system to execute Natural programs.

Object

In the object-oriented programming model, data structures and functions (so-called *methods*) are packaged together in objects. Each object belongs to a class, which describes the internal structure of the object and its interfaces, properties and methods.

Object Data Area - ODA

The object data area is the place where the current values of all properties of an object are stored. Also other variables can be defined in the object data area, which are not accessible by clients as properties, but just used by the methods of the object to maintain an internal state of the object. The structure of the object data area of all objects of one class is specified in the OBJECT USING clause in the DEFINE CLASS statement. An object data area is nothing else than a local data area, and in fact it is also created in the data area editor as a local data area.

Object Data Variable

Each property needs a variable in the object data area of the class to store its value - this is referred to as the object data variable.

ProgID

The ProgID (programmatic identifier) of a DCOM class is a meaningful name by which the class is identified in client programs. For Natural classes, the name defined in the *class-name* operand in the DEFINE CLASS statement is written into the registry as a ProgID when the class is registered as a DCOM class with the REGISTER command.

Property

Properties are attributes of an object that can be accessed by clients. In Natural classes, property values of an object are stored in the object data area. Therefore an object data variable must be assigned to each property.

For further information, see the section Object Data Variable.

Registry

A repository containing operating system information. The registry also contains information about DCOM classes and their assignment to servers.

Registry Key

Registry keys are entries made in the system registry of the server when a class is registered. Registry keys are also added in the client system registry when the client registration file is executed.

For detailed background information about the registry keys and their administration, please refer to the registry documentation of the appropriate platform.

Server ID

A server ID is a character string that identifies a NaturalX server. The server ID is a Natural-owned key in the system registry, keeping together all classes that belong to a given NaturalX server. It is an arbitrary alphanumeric string of 32 characters which does not contain blanks and which is not case sensitive. The server ID is defined with the Natural parameter `COMSERVERID=serverid` (for OS/390, `DCOM=(SERVID=serverid)`).

Type Information

When a Natural class is registered as a DCOM class, a type library is generated for the class and connected to the class by a registry entry. Clients can use the type information contained in the type library to check the descriptions of the interfaces, methods and properties of the class either at compile time or at runtime.

Type Library

When a Natural class is registered as a DCOM class, a type library is generated for the class and connected to the class by a registry entry. Clients can use the type information contained in the type library to check the descriptions of the interfaces, methods and properties of the class either at compile time or at runtime.